POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

PhD Thesis

STAGED SELF-SUPERVISION FOR SOLVING RAVEN'S PROGRESSIVE MATRICES

Jakub Kwiatkowski

Supervisor
prof. dr hab. inż. Krzysztof Krawiec

POZNAŃ 2026

Abstract

Solving visual reasoning tasks requires more than recognizing surface patterns: it demands the capacity to discover and apply abstract relational rules across novel configurations. Raven's Progressive Matrices (RPMs)—visual puzzles in which a missing panel must be inferred to complete a structured geometric pattern—have become a standard benchmark for assessing such capabilities in machine learning systems. Contemporary approaches predominantly rely on monolithic architectures trained to select the correct answer from a fixed candidate pool. This formulation conflates perception, reasoning, and decision-making into a single undifferentiated process, limiting the model's ability to acquire structured representations of the underlying rules and creating systematic vulnerabilities to statistical regularities in the answer set—a class of failure modes known as shortcut learning.

This thesis proposes a staged self-supervision framework that reformulates RPM solving by decomposing it into two distinct stages: attribute prediction and choice making. The first stage employs the Abstract Compositional Transformer (ACT), a transformer-based architecture that learns to infer structured, symbolic descriptions of RPM panels by predicting masked entries from the surrounding context. The second stage introduces three alternative choice-making modules—the Direct Choice Maker, the Learned Choice Maker, and the Contrastive Choice Maker—each comparing the predicted attributes with those of answer panels to identify the correct solution.

The framework achieves results competitive with the strongest approaches reported on the RAVEN, I-RAVEN, and RAVEN-FAIR benchmarks, without requiring dataset-specific retraining for each. The staged decomposition additionally mitigates dataset biases by architectural design: the attribute prediction stage operates exclusively on context panels without answer set exposure, while each choice-making module evaluates candidates independently, eliminating pathways for statistical exploitation. Extensive empirical analysis—including ablation studies, data efficiency experiments, and visualization of learned representations—confirms that the framework acquires compositional structure rather than surface-level correlations. The work establishes that principled decomposition combined with self-supervised learning enables neural networks to discover structured representations of abstract visual rules.

Keywords: abstract reasoning, Raven's Progressive Matrices, self-supervised learning, transformer architectures, shortcut learning, representation learning

Streszczenie

Skuteczne rozwiązywanie zadań wymagających rozumowania wizualnego nie polega jedynie na rozpoznawaniu powierzchownych wzorców. Wymaga uchwycenia abstrakcyjnych zależności między elementami oraz zastosowania ich w nowych konfiguracjach. Progresywne Matryce Ravena (RPM) — wizualne zagadki, w których należy wskazać brakujący panel uzupełniający wzorzec — stały się powszechnie stosowanym benchmarkiem służącym do oceny takich zdolności w systemach uczenia maszynowego. Dominujące podejścia opierają się na monolitycznych architekturach uczonych do wskazywania poprawnej odpowiedzi spośród ustalonego zbioru kandydatów. Takie ujęcie zadania zaciera granice między percepcją, rozumowaniem i podejmowaniem decyzji, przez co utrudnia modelowi nabywanie reprezentacji reguł rządzących wzorcem. Jednocześnie sprzyja wykorzystywaniu statystycznych regularności obecnych w zbiorze odpowiedzi zamiast rzeczywistego rozumowania — zjawisko to określa się mianem uczenia na skrótach (*shortcut learning*).

Niniejsza praca proponuje metodę samonadzorowanego uczenia etapowego (*staged self-supervision*), która przeformułowuje problem rozwiązywania RPM jako dwuetapowy proces obejmujący predykcję atrybutów oraz wybór odpowiedzi. W pierwszym etapie zastosowano Abstract Compositional Transformer (ACT) — architekturę transformerową uczącą się przewidywać symboliczne opisy paneli RPM poprzez rekonstrukcję zamaskowanych paneli na podstawie otaczającego kontekstu. Drugi etap realizują trzy alternatywne moduły wyboru odpowiedzi: *Direct Choice Maker*, *Learned Choice Maker* oraz *Contrastive Choice Maker*, z których każdy porównuje przewidywane atrybuty z atrybutami paneli kandydujących i wskazuje rozwiązanie.

Zaproponowana metoda osiąga wyniki konkurencyjne wobec najlepszych opublikowanych rozwiązań na benchmarkach RAVEN, I-RAVEN oraz RAVEN-FAIR, bez konieczności odrębnego trenowania dla każdego z nich. Dwuetapowy proces ogranicza ponadto wpływ obciążeń zbioru danych już na poziomie architektury: etap predykcji atrybutów operuje wyłącznie na panelach kontekstowych, natomiast każdy moduł ocenia kandydatów niezależnie, uniemożliwiając eksploatację statystycznych regularności zbioru odpowiedzi. Obszerna analiza empiryczna — obejmująca badania ablacyjne, eksperymenty dotyczące efektywności danych oraz wizualizację nauczonych reprezentacji — potwierdza, że metoda uczy się reguł rządzących wzorcem, a nie jedynie powierzchownych zależności statystycznych. Wyniki wskazują, że przemyślana dekompozycja zadania połączona z samonadzorowanym uczeniem pozwala sieciom neuronowym

skutecznie rozwiązywać zadania abstrakcyjnego rozumowania.

Słowa kluczowe: rozumowanie abstrakcyjne, Progresywne Matryce Ravena, samonadzorowane uczenie, architektury transformerowe, uczenie na skrótach (*shortcut learning*), uczenie reprezentacji

Contents

Chapter 1

Introduction

1.1 Context and Motivation

Artificial General Intelligence (AGI) represents the pinnacle of artificial intelligence, aspiring to create machines that possess human-like cognitive abilities, enabling them to understand, learn, and adapt to a wide range of tasks and environments. The goal of AGI is to develop systems capable of performing any intellectual task that a human can, with the same level of flexibility, creativity, and problem-solving skills (Chollet, 2019). One of the key attributes of general intelligence is *abstract reasoning*, which, among others, subsumes the capacity to reason about and complete sequential patterns. Abstract reasoning encompasses abilities such as identifying patterns, principles, analogies, and relationships, as well as logical thinking. At a higher level, it involves the ability to conceptualize, which includes understanding and applying abstract and complex concepts, and manipulating and reasoning about them.

To quantify such capabilities in humans and diagnose related deficiencies, John C. Raven devised a visual test in the 1930s, now commonly known as **Raven's Progressive Matrices (RPMs)** (Raven, 1936). An RPM problem (Figure 1.1) consists of a 3×3 grid of eight context panels containing arrangements of 2D geometric objects. The objects adhere to rules that govern the relationships between the panels, e.g., progression of the number of objects, a logical operation concerning their presence, etc. The task is to complete the puzzle by replacing the lower-right query panel with the correct image from the eight provided answer panels. Solving the task requires disentangling the rules corresponding to rows and columns and capturing the analogies between the observed patterns.

RPMs have been more recently adopted in the AI community for assessing similar capabilities in artificial intelligence systems (Zhang et al., 2019a; Małkiński and Mańdziuk, 2023), along with other benchmarks like Bongard problems (Bongard, 1970; Nie et al., 2020) and Hofstadter's analogies (Hofstadter, 1995). Recent advances in machine learning have accelerated this trend, with deep learning becoming the dominant paradigm for designing such systems (Małkiński and Mańdziuk, 2025). The original collection of RPM problems comprised only 60 tasks, which proved insufficient for efficiently training

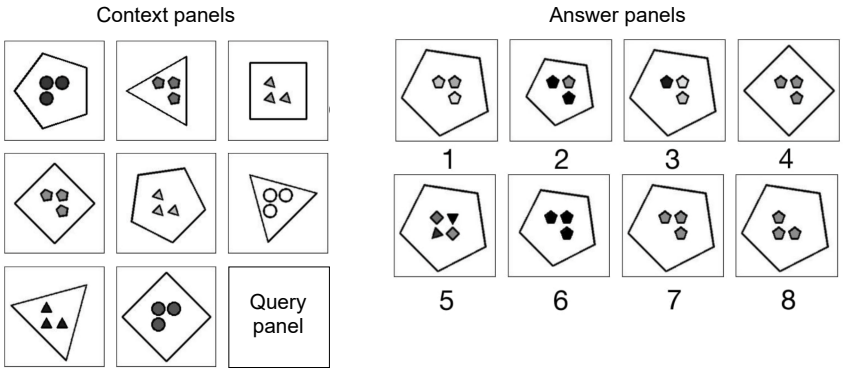


FIGURE 1.1: An RPM task: choose the *answer panel* which, when placed in the *query panel* location, completes the patterns observed in the *context panels*.

data-hungry machine learning models. Consequently, researchers have developed larger datasets and task generators, with **RAVEN** (Zhang et al., 2019a) and **I-RAVEN** (Hu et al., 2021) being among the most popular nowadays.

Contemporary deep learning often proves proficient at building end-to-end models that map raw input data to the desired output, where intermediate representations learn to align with the characteristics of the task. However, tasks that involve mapping a high-dimensional input to a low-dimensional output are usually underconstrained and make overfitting to spurious or irrelevant features more likely, especially for heavily overparameterized models, which prevail in deep learning (Hastie et al., 2009). The risk of intermediate representations becoming misaligned with the task is elevated when the training data are scarce or in some sense imperfect. One category of problems that is particularly prone to both these factors is abstract reasoning tasks (Zhang et al., 2019a; Spratley et al., 2020).

Motivation: Identifying the Gap. Contemporary deep learning systems face a fundamental challenge: they are inherently designed to find the simplest solution that maximizes performance within their network capacity, gravitating toward statistical shortcuts rather than methodical symbolic reasoning (Geirhos et al., 2020). This phenomenon, known as **shortcut learning**, manifests across diverse domains: in visual question answering, models often exploit language biases and ignore visual content (Wen et al., 2021), while recent research on large language models has revealed reliance on pattern matching rather than step-by-step logical reasoning when solving mathematical problems (Mirzadeh et al., 2024).

This limitation becomes especially problematic in abstract reasoning tasks like RPMs, where models are expected to perform logical inference rather than pattern matching. RPM solvers can often learn by observing answer panels alongside context panels and are trained to choose the correct answer panel in a supervised fashion. However, this method creates multiple pathways for shortcut learning that undermine the intended reasoning process.

The RAVEN benchmark is systematically affected by biases that leak the correct answer through the distribution of features in answer panels (Spratley et al., 2020). The distractor panels are constructed to closely resemble the correct answer panel, which might suggest the task is difficult, but this very regularity means the correct answer can often be identified as the one whose features occur most frequently among all answer panels. This characteristic, known as **answer set bias**, makes it possible to attain almost perfect performance by learning from the answer panels alone and entirely ignoring the context panels. Additionally, another bias we term **similarity bias** allows models to gain an advantage by selecting the answer panel that is most similar to the context panels, rather than discovering the logical rule governing the pattern succession.

This is why we argue that traditional choice-based evaluation serves as an inadequate benchmark for assessing a model's ability to reason. *A much better approach would be to require the model to*

construct the answer, not just select it. Instead of mere selection, this thesis proposes constructing the correct answer by predicting the query panel properties that allow reconstruction of the panel. We postulate that open-ended tasks, where the algorithm needs to construct an answer, provide a much better estimate of an algorithm's cognitive abilities, as construction tasks are less prone to bias than those mandating selection. This motivates our staged decomposition strategy, which explicitly separates property prediction from answer selection, thereby addressing the shortcut learning tendencies of deep learning systems by combining representation learning with structured decision-making processes.

1.2 Aims, Scope, and Hypotheses

The principal objective of this thesis is to develop, implement, and evaluate a novel framework for solving RPMs that promotes abstract reasoning and is resilient to shortcut learning and dataset biases. Rather than training models that choose answer panels in RPMs, we propose to rely on *property prediction*, in which models construct an abstract, structured representation of the missing panel. To this end, we decompose RPM tasks into two stages: (i) prediction of properties of the query panel, and (ii) identification of the answer panel with properties that match the predicted ones the best.

We introduce the **staged self-supervision methodology**, which combines an **Abstract Compositional Transformer (ACT)** for property prediction with various **choice-making modules** for answer selection. The Abstract Compositional Transformer is designed to predict the symbolic properties of RPM panels using self-supervised learning, while the choice-making module (such as Direct Choice Maker, Learned Choice Maker, and Contrastive Choice Maker) compares these predictions with the properties of answer panels to make the final selection.

Scope. We evaluate the framework across multiple benchmarks, with primary evaluation on **RAVEN** (Zhang et al., 2019a) and **I-RAVEN** (Hu et al., 2021), examining accuracy, robustness to dataset biases, and interpretability of the learned symbolic representations of panels. While our empirical work centers on Raven's Progressive Matrices, the principles underlying our strategy—particularly the separation of property prediction from choice making and the reliance on self-supervised learning—suggest broader applicability beyond the RPM domain. We explore these connections in Chapter 6, discussing how the methodology might extend to tasks involving analogical reasoning, pattern completion, and visual concept learning.

Research Hypotheses. This dissertation is structured to validate three central hypotheses that form the argumentative backbone of our approach:

H1 Stage-wise decomposition of abstract reasoning tasks facilitates obtaining better solvers. We hypothesize that breaking down the RPM solving task into two distinct stages—property prediction and choice making—leads to more effective and interpretable solutions compared to monolithic approaches. This decomposition allows each stage to focus on its specific sub-task, leading to improved overall performance.

H2 Delineating property prediction as a sub-task helps in mitigating biases present in RPM benchmarks. Constructing entirely unbiased answer sets remains a persistent challenge in RPM benchmark design, as the very nature of multiple-choice formulations creates opportunities for statistical exploitation. We propose that by decomposing the task into property prediction and choice making, we can substantially limit the model's exposure to answer sets and thereby mitigate their biasing influence.

H3 Self-supervision is an adequate training regime for building RPM solvers. We hypothesize that self-supervised learning, where the model learns to predict masked panel properties, is suffi-

cient for training competent RPM solvers. This approach enables the model to learn meaningful representations from the structural information available in RPM tasks.

1.3 Summary of Contributions

This thesis presents several scientific and engineering contributions to the field of abstract visual reasoning:

1. **Novel Two-Stage Framework**: We introduce the **staged self-supervision methodology** for abstract reasoning that decouples property prediction from answer selection. This framework addresses the structural limitations of monolithic approaches by eliminating exposure to answer set biases during the critical learning phase.

2. **Abstract Compositional Transformer (ACT)**: We design and implement a transformer-based architecture tailored to predicting symbolic representations of RPM panels. The ACT model learns to predict abstract properties such as shape, color, size, and spatial arrangements through self-supervised learning on masked panel sequences.

3. **Choice-Making Modules**: We develop multiple approaches to choice making, including Direct Choice Maker (DCM), Learned Choice Maker (LCM), and Contrastive Choice Maker (CCM), each offering different strategies for comparing predicted properties with answer panel properties to select the correct solution.

4. **Competitive Performance**: Our framework achieves results among the strongest reported on both RAVEN and I-RAVEN benchmarks, attesting to the strength of the staged design.

5. **Bias Resistance**: We provide empirical evidence that our framework exhibits robustness to dataset biases. By requiring the model to reason about context panels rather than exploiting answer set statistics, our design structurally limits the pathways available for shortcut learning.

6. **Self-Supervised Learning for Abstract Reasoning**: We demonstrate that self-supervised learning is a viable and effective training regime for RPM solving, showing that successful learning can occur using the structural information present in the dataset.

7. **Interpretable Symbolic Representations**: Our model produces *interpretable symbolic representations* of panel properties, offering insights into the reasoning process that are unavailable in black-box end-to-end systems. This transparency facilitates analysis of model behavior and failure modes.

8. **Comprehensive Empirical Analysis**: We provide extensive experimental validation including ablation studies, scalability analysis, and detailed examination of learned representations, contributing to a deeper understanding of what makes RPM solvers capable.

The core content of this thesis has been covered in the following publications and poster:

Journal and conference publications:

- **Staged Self-Supervised Learning for Raven Progressive Matrices** (2025)
  Jakub Kwiatkowski, Krzysztof Krawiec
  IEEE Transactions on Neural Networks and Learning Systems, vol. 36, no. 9, pp. 16564–16576
  Impact Points: 200
  `https://ieeexplore.ieee.org/abstract/document/11014503`

- **Learning Abstract Visual Reasoning via Task Decomposition: A Case Study in Raven Progressive Matrices** (2024)
  Jakub Kwiatkowski, Krzysztof Krawiec
  International Journal of Applied Mathematics and Computer Science, vol. 34, no. 2, pp. 309–321
  Impact Points: 100
  `https://doi.org/10.61822/amcs-2024-0022`

Posters and presentations:

- **Self-supervised Learning of Tokenized Representations for Raven Progressive Matrices** (2024)
  Jakub Kwiatkowski, Krzysztof Krawiec
  Poster presentation at *PP-RAI 2024 — 5th Polish Conference on Artificial Intelligence*, 18–20 April 2024, Warsaw University of Technology, Warsaw, Poland
  `https://pprai24.mini.pw.edu.pl/en/general-information/conference-program.html`

1.4 Thesis Outline

This dissertation is organized as follows:

Chapter 1: Introduction motivates the study of abstract visual reasoning and identifies the limitations of monolithic end-to-end approaches to RPM solving. It defines the research hypotheses, outlines the proposed staged self-supervision framework, and summarizes the main contributions of this thesis.

Chapter 2: Background establishes the theoretical foundation for this research. It provides a comprehensive review of abstract reasoning in artificial intelligence, examines the advantages and limitations of deep learning, and positions this thesis within the broader context of solving reasoning tasks.

Chapter 3: Raven's Progressive Matrices as an Abstract Reasoning Task provides a detailed examination of the benchmarks used in this thesis. It focuses on Raven's Progressive Matrices (RPMs), detailing the structure of the RAVEN and I-RAVEN datasets. A portion of this chapter is dedicated to analyzing the inherent dataset biases, such as answer set bias and similarity bias, which challenge existing methods.

Chapter 4: Abstract Compositional Transformer introduces the first core component of our methodology. This chapter details the architecture of the Abstract Compositional Transformer (ACT), our novel model for property prediction. It covers the tokenization strategies, the self-supervised training objective, and the mechanisms by which the model learns to predict structured, symbolic representations of panel properties.

Chapter 5: Choice Makers describes the second stage of our framework. It presents the various choice-making modules—Direct (DCM), Learned (LCM), and Contrastive (CCM)—that use the predictions from the ACT to select the correct answer. This chapter details their implementation and presents the experimental results that validate their utility.

Chapter 6: Additional Investigations and Future Work extends the evaluation beyond the standard RAVEN benchmark, testing the framework on procedurally generated matrices (PGM) and exploring its generalization capabilities. It also presents an investigation of large language models as abstract reasoners and discusses promising directions for future research.

Chapter 7: Summary and Conclusions synthesizes the key findings of the thesis and evaluates the final status of our research hypotheses. It discusses the broader implications of staged self-supervision for abstract reasoning, acknowledges limitations, and outlines promising directions for future research.

Appendix contains supplementary material, including detailed experimental results, and additional analyses that support the main chapters.

1.5 PhD Candidate Profile

The author of this thesis is a PhD student in Computer Science at Poznan University of Technology, with primary research interests centered on **abstract reasoning** in deep neural networks, **representation learning**, and **explainable AI**. This academic pursuit is complemented by professional expertise gained through employment at the Poznan Supercomputing and Networking Center, where the author serves as an AI Engineer specializing in the application of deep learning algorithms to cybersecurity challenges and AI system security.

The author's research expertise extends to identifying failure modes in large language models and machine learning systems, focusing on demonstrating fundamental limitations in abstract reasoning capabilities. This pursuit involves studying **shortcut learning patterns**—where models rely on superficial correlations rather than genuine understanding—and employing adversarial techniques including **prompt injections** and **jailbreaking attacks** across various model architectures.

In his professional capacity, the author works as an auditor of production-grade AI systems, evaluating deployed artificial intelligence solutions for security compliance, bias assessment, and operational risk mitigation. Additionally, the author contributes to the AI security community through training initiatives focused on securing artificial intelligence systems, addressing challenges ranging from adversarial robustness to ethical deployment considerations. This line of work requires comprehensive research encompassing AI lifecycle threats, including analysis of poisoned datasets, compromised pre-trained models, and tampered training pipelines, with particular attention to mitigating stealthy attack methods that evade traditional detection.

This combination of academic research in abstract reasoning and practical experience in AI security provides a distinctive perspective that informs the research methodology and contributions presented in this thesis, particularly in developing trustworthy AI systems capable of robust and interpretable abstract visual reasoning.

Journal and conference publications:

- **Estimating the vulnerability of industrial network infrastructure in Central and Eastern Europe** (2024)
  Mateusz Twardawa, Marek Smolik, Franciszek Rakowski, Jakub Kwiatkowski, Norbert Meyer
  Security and Defence Quarterly, vol. 47, no. 3, pp. 53–73
  Impact Points: 70
  `https://www.ceeol.com/search/article-detail?id=1272933`

- **SCADvanceXP – an intelligent Polish system for threat detection and monitoring of industrial networks** (2024)
  Mateusz Twardawa, Marek Smolik, Franciszek Rakowski, Jakub Kwiatkowski, Norbert Meyer
  Security and Defence Quarterly, vol. 48, no. 4, pp. 19–39
  Impact Points: 70
  `https://www.ceeol.com/search/article-detail?id=1323495`

- **Explainable Hybrid CNN and FNN Approach Applied on Robotic Wall-Following Behaviour Learning** (2021)
  Jakub Kwiatkowski, Liang Ou, Yu-Cheng Chang, Chin-Teng Lin
  In: *AIPR 2021: 4th International Conference on Artificial Intelligence and Pattern Recognition*,

17–19 September 2021, Xiamen, China: ACM, 2021, pp. 623–628
Impact Points: 20
https://dl.acm.org/doi/abs/10.1145/3488933.3489026

Posters and presentations:

- **Overview of AI Security Landscape** (2025)
  Jakub Kwiatkowski, Maksymilian Marcinowski
  Presentation at *GÉANT Security Days 2025 — "Securing What Matters"*, 9 April 2025, Grandior Hotel, Prague, Czech Republic
  https://security.geant.org/geant-security-days-2025/

- **Systemy sztucznej inteligencji jako nowy obszar cyberzagrożeń** (2026)
  Jakub Kwiatkowski, Maksymilian Marcinowski
  Presentation at *I Poznańska Konferencja Kryminalistyczna*, 12 March 2026, Collegium Iuridicum Novum, Adam Mickiewicz University, Poznań, Poland
  https://poz-krym.web.amu.edu.pl/

- **Honeypots and beyond – LLMs in cyberdefence** (2026)
  Maciej Miłostan, Michał Ślusarczyk, Jakub Kwiatkowski, Maksymilian Marcinowski, Mikołaj Dobski
  Presentation at *TNC26*, session "To Serve and Protect", 9 June 2026, Musiikkitalo, Helsinki, Finland
  https://tnc26.geant.org/programme/#Tuesday/all/s1068

- **Your AI Blind Spot: The Attack Surface You Didn't Know You Had** (2026)
  Jakub Kwiatkowski
  Lightning talk at *TNC26*, session "Lightning Talks: Strike One", 10 June 2026, Musiikkitalo, Helsinki, Finland
  https://tnc26.geant.org/programme/#Wednesday/all/s1076

Chapter 2

Background

2.1 Foundations of Modern AI

2.1.1 Deep Learning's Remarkable Achievements

Deep neural networks excel in learning representations from raw data, allowing them to automatically extract features and patterns without the need for manual feature engineering. As a result, deep learning has achieved remarkable success in tasks such as image recognition (He et al., 2016), natural language processing (Devlin et al., 2018), and speech recognition (Hinton et al., 2012), demonstrating its potential to revolutionize numerous industries and applications.

Deep learning is becoming increasingly intertwined with our lives and finds applications in a diverse array of fields, even beyond traditional IT domains. It plays a crucial role in weather forecasting, where models like Convolutional LSTM Networks (Xingjian et al., 2015) and MetNet (Sønderby et al., 2020) are utilized to predict precipitation patterns with increasing accuracy. Recent advancements in magnetic control systems demonstrate how deep learning techniques assist in regulating processes within tokamaks, advancing research in the field of nuclear fusion (Degrave et al., 2022). Deep learning algorithms contribute to the restoration and attribution of ancient texts (Assael et al., 2022), marking a significant advancement in historical preservation. In astronomy, machine learning methods are leveraged to sift through vast amounts of radio telescope data, enabling the detection of anomalous signals indicative of potential technosignatures (Ma et al., 2023).

These applications underscore the versatility and impact of deep learning across various scientific and practical domains, extending its reach beyond traditional boundaries. The success of deep learning in these diverse areas stems from multiple complementary factors, including its ability to learn **hierarchical representations** that capture increasingly abstract features at different levels of processing. **Hierarchical learning** represents one important mechanism among several that enable neural networks to handle complex tasks (Chen et al., 2020a). Such hierarchical decomposition of feature learning provides

theoretical support for the *staged decomposition approaches* explored in this thesis, where advanced reasoning tasks are decomposed into explicit stages that enable progressive learning of increasingly high-level concepts. This decomposition allows each stage to focus on learning representations at the appropriate level of abstraction, with each stage constrained to learn the representations most critical for its specific role in the overall task. In our abstract reasoning case, the first stage is constrained to learn symbolic property representations that are most critical for abstract reasoning, while the second stage specializes in decision-making logic. This targeted learning approach potentially improves both interpretability and generalization compared to monolithic end-to-end approaches that learn all representational levels simultaneously, without explicit guidance on which abstractions are most critical at each processing stage.

However, this staged approach introduces an important tension: explicit decomposition into different components constrains the solution space and potentially limits the paths available to reach optimal solutions. If unconstrained, fully continuous deep learning already learns representations that, while perhaps not perfect, hierarchically discover new concepts and features through end-to-end optimization. This raises a question: do we truly need additional architectural constraints that potentially restrict the learning procedure, or might such constraints inadvertently prevent the discovery of superior implicit representations that could emerge from unconstrained learning?

This tension between explicit structural guidance and unconstrained representational discovery exemplifies the *bias-variance trade-off* in machine learning (Hastie et al., 2009). Whether imposing explicit structural constraints substantively improves systematic reasoning and bias-resistance, or whether it instead restricts the solution space in ways that ultimately limit performance, is a central design choice examined empirically in the chapters that follow.

2.1.2 Bias-Variance Trade-Off

The bias-variance trade-off revolves around achieving a balance between a model's simplicity (bias)—the degree to which it is constrained to a restricted hypothesis space—and its ability to accurately capture underlying patterns (variance). Models exhibiting high bias tend to be overly constrained, unable to capture the intricacies present in the data, leading to underfitting. Conversely, models characterized by high variance tend to be excessively complex, capturing noise along with genuine patterns and resulting in overfitting.

In the context of deep learning, this trade-off is particularly relevant due to the non-linear nature of deep neural networks. Deep learning models tend to have high variance, meaning they can fit the training data very closely, but this may lead to overfitting and poor generalization to unseen data (Hastie et al., 2009). This challenge becomes especially pronounced in abstract reasoning tasks, where models must learn to generalize beyond superficial statistical patterns to capture underlying logical structures.

To mitigate overfitting, various constraints and biases are introduced into the models, including regularization techniques and specialized architectures. These biases and constraints exist in diverse forms, including regularization techniques like dropout (Srivastava et al., 2014) and weight decay, aimed at encouraging the model to extract more general features; convolutional layers (LeCun et al., 1998) assuming spatial relationships within input data; recurrent neural networks (Hochreiter and Schmidhuber, 1997) accounting for the sequential nature of data; and transformers (Vaswani et al., 2017) modeling token co-dependency.

These measures collectively guide models in suitable learning directions, harnessing the vast capacity of deep neural networks to learn efficiently from data. This intricate balance allows deep learning to emerge as a dominant paradigm in artificial intelligence, offering unparalleled flexibility and scalability for solving multi-faceted tasks across various domains. As we will demonstrate throughout this thesis, **H1** (Sec. 1.2)—that stage-wise decomposition facilitates better solvers—provides a principled approach to

achieving this balance by explicitly structuring the learning process into hierarchical stages, particularly in abstract reasoning tasks such as **Raven's Progressive Matrices (RPMs)**, where systematic reasoning is crucial for consistent performance.

2.2 The Scaling Paradigm: Promise and Fundamental Limitations

2.2.1 Neural Scaling Laws and Their Foundations

The bias-variance trade-off discussed in the previous section (Sec. 2.1.2) reveals a significant cost of deep learning's flexibility: dependence on vast amounts of data. **Neural scaling laws** (Kaplan et al., 2020) formalize this relationship, describing how model performance follows power-law distributions relative to data volume, parameter count, and computational resources. The implications are sobering: even modest performance improvements may demand substantially disproportionate increases in training data.

Refining this picture, Hoffmann et al. (2022) demonstrate that for a given compute budget, optimal performance emerges from training smaller models on more data rather than larger models on less data. This finding challenges the prevalent practice of scaling parameters while keeping datasets fixed and suggests that the quality and curation of training data may be more impactful than sheer quantity.

Building on these insights, investigations have revealed *emergent abilities* (Wei et al., 2022a) in large models—capabilities that appear suddenly at certain scales rather than improving gradually. These phenomena include few-shot learning, chain-of-thought reasoning, and instruction following. However, the emergence of these abilities does not necessarily indicate understanding or systematic reasoning capabilities. Critical analyses suggest that these apparent emergent abilities may represent sophisticated illusions rather than authentic cognitive breakthroughs. Schaeffer et al. (2023) demonstrate that alleged emergent abilities can "evaporate with different metrics or with better statistics," suggesting they may result from researchers' choice of evaluation metrics rather than fundamental changes in model behavior. The authors argue that nonlinear or discontinuous metrics may produce apparent emergent abilities, whereas linear or continuous metrics could reveal smooth, continuous, and predictable changes in model performance. This indicates that what appears to be sudden capability emergence may actually reflect models reaching sufficient scale to convincingly simulate reasoning through sophisticated pattern recognition and statistical learning. Berti et al. (2025) argue that neural networks have achieved a level of *interpolation sophistication* that enables them to simulate emergent reasoning behaviors without developing the underlying **systematic reasoning capabilities** that would characterize actual intelligence.

2.2.2 The Scaling vs. Novel Components Debate

Although neural scaling laws have sparked **divergent perspectives** among researchers regarding their universal applicability and underlying mechanisms, with some research suggesting potential deviations from power law relationships through selective data curation (Caballero et al., 2022) and others questioning the genuine nature of apparent emergent abilities (Schaeffer et al., 2023), the **scaling paradigm** continues to dominate the field of AI research (Hassabis and Patel, 2024). Scaling approaches—through increasing data volumes, model parameters, and computational resources—have consistently delivered practical breakthroughs and performance gains across diverse domains (Kaplan et al., 2020; Brown et al., 2020). This empirical success, even amid ongoing controversy, has sparked fundamental debate about whether achieving artificial general intelligence requires simply scaling existing methods or developing novel **architectural components** with additional inductive biases (Marcus, 2020; Mitchell, 2021). This debate has profound implications for the future direction of AI research and the approaches explored in this thesis.

The Case for Scaling: Sutton's Bitter Lesson as Foundational Idea

A substantial and influential argument supporting the scaling approach comes from Richard Sutton's essay "The Bitter Lesson" (Sutton, 2019). Sutton argues that *general-purpose learning modules* that can easily and continuously scale with computational resources consistently outperform *specialized modules* designed for specific tasks. The lesson emphasizes that researchers consistently underestimate the power of general methods that leverage computation, while the integration of human knowledge often complicates systems and makes them less suited for exploiting computational power.

This perspective has profoundly influenced AI research, leading to the current emphasis on scaling transformer-based models and the belief that artificial intelligence emerges through computational scale rather than architectural innovation. The remarkable success of this approach has led some researchers to confidently assert that we now possess all essential tools for developing artificial general intelligence (AGI), with scaling as the only remaining step (de Freitas, 2022). The underlying assumption is that cognitive functions such as reasoning will naturally emerge as inherent properties of larger models trained on extensive datasets.

This seemingly simplistic perspective gains credibility from the empirical reality that most recent breakthroughs have indeed stemmed from scaling existing methods rather than introducing fundamentally novel techniques (Kaplan et al., 2020; Hoffmann et al., 2022; Hassabis and Patel, 2024). Transformers exemplify this trend, representing arguably more general computational frameworks than previous architectures such as feedforward layers and convolutional networks. Unlike traditional architectures that multiply input data by fixed learned weights, attention mechanisms enable *input-dependent* computations: tokens dynamically interact with one another through content-driven transformations rather than static, position-fixed operations. This allows attention to relate any piece of information to any other piece in the sequence without inherent *spatial or sequential biases*, a property absent from both convolutional and recurrent architectures. Moreover, the transformer architecture's design—with multiple transformer blocks, multiple attention heads, and position-independent self-attention computations that eliminate sequential dependencies—enables exceptional parallelization that facilitates efficient scaling across computational resources (Vaswani et al., 2017).

This architectural advantage has enabled transformers to achieve unprecedented performance across diverse domains—from natural language processing to computer vision to speech recognition— primarily through scale rather than domain-specific innovations (Brown et al., 2020; Dosovitskiy et al., 2020; Radford et al., 2021). Contemporary developments in transformer architectures, including models like PaLM (Chowdhery et al., 2022) and LaMDA (Thoppilan et al., 2022), along with more recent innovations such as DeepSeek-V3 (DeepSeek-AI, 2024) and Grouped Query Attention (GQA) (Ainslie et al., 2023), represent iterative enhancements of the foundational transformer design. These advancements continue to focus primarily on scaling strategies, optimization procedures, and architectural refinements, such as Multi-Head Latent Attention (MLA) (DeepSeek-AI, 2024), better parallelization strategies (Chowdhery et al., 2022), and enhanced efficiency techniques through attention optimization (Dao et al., 2022), rather than introducing fundamentally new computational paradigms.

2.2.3 Limitations of Pure Scaling

Limitations of Pure Scaling in Abstract Reasoning

While the scaling paradigm discussed in the previous section (Sec. 2.2.1) remains the main driver of current advances in AI and keeps delivering large gains across diverse domains, it faces inherent limitations when applied to **abstract reasoning** tasks.

Abstract reasoning necessitates the capacity to **infer and systematically apply** complex rules to novel configurations (Sternberg, 1985; Holland et al., 1986). Unlike pattern recognition, where statistical regularities often suffice, abstract reasoning demands extracting **symbolic relationships** and applying them consistently across varying contexts (Lake et al., 2017). However, scaling-centric models tend to rely on memorization and interpolation within training distributions, falling short when generalizing to unseen combinations of rules and entities (Lake and Baroni, 2018; Bahdanau et al., 2018). Empirical studies reveal that even large models struggle with rigorous rule-based reasoning, often relying on superficial heuristics rather than discovering underlying principles (Dziri et al., 2023; Valmeekam et al., 2022; McCoy et al., 2019).

These limitations manifest across multiple domains. In algorithmic reasoning, neural networks trained on sorting algorithms demonstrate severe performance degradation when list sizes exceed those seen during training (Graves et al., 2014). Similarly, models struggle with **compositional generalization**, where they must recombine known components in novel ways. The SCAN dataset (Lake and Baroni, 2018) exemplifies this challenge: it contains navigation commands combining primitive actions (*jump*, *run*, *walk*) with modifiers (*twice*, *opposite*, *left*, *around right*), requiring models to translate commands like `jump left twice` into action sequences such as `LTURN JUMP LTURN JUMP`. Recent studies reveal that models trained on the SCAN benchmark fail catastrophically when tested on novel constructions requiring systematic recombination of known components. For instance, in the most challenging experimental setup, models are trained on the primitive command *jump* alongside all composed commands involving other actions (such as *run twice*, *walk opposite left*, and *walk around right thrice*). However, during testing, models must execute composed commands involving *jump* that they have never encountered, such as *jump twice* and *jump opposite left and run twice*. The results reveal a complete breakdown in systematic generalization: the best-performing model on jump-related compositional commands achieves only 1.2% accuracy, while the overall best model across all commands reaches a dismal 0.08% accuracy on jump compositions (Lake and Baroni, 2018).

Studies by Kim et al. (2018) and Ricci et al. (2018) have exposed substantial weaknesses of feedforward neural networks in learning and generalizing abstract relational concepts, particularly the notion of *sameness* in visual reasoning tasks. The *same-different* relation requires determining whether two items are identical regardless of specific visual attributes—a form of relational abstraction representing a fundamental cognitive capability. However, neural networks consistently fail this generalization even under controlled experimental conditions. In experiments with a same-different variant of the sort-of-CLEVR dataset (Santoro et al., 2017; Kim et al., 2018), models were trained to recognize the sameness relation using color-shape combinations (e.g., red circles, blue squares), where one particular combination (e.g., cyan squares) was withheld during training and reserved for testing. Although the models observed each individual attribute (cyan color and square shape) separately during training, they failed catastrophically when required to generalize the same-different relation to the novel combination, as illustrated in Fig. 2.1. This failure reveals that networks memorize specific color-shape co-occurrences rather than the underlying relational rule—that two objects are *same* if and only if they match on every attribute jointly. Notably, objects in these images are placed at random positions on the background, so position varies freely; the failure is purely compositional, not spatial. The limitation extends beyond feature combinations to spatial configurations in other *same-different* datasets such as *PSVRT* (Ricci et al., 2018): even with only two objects per image, CNNs fail to learn a general comparison rule and instead memorize position-specific templates, with performance degrading as the image grows larger and the number of possible arrangements increases.

These failures illustrate a recurring pattern: monolithic networks, trained end-to-end on a single objective, tend to exploit the simplest available statistical signal rather than developing structured relational representations (Marcus, 2018). This provides compelling motivation for **H1** (Sec. 1.2): that

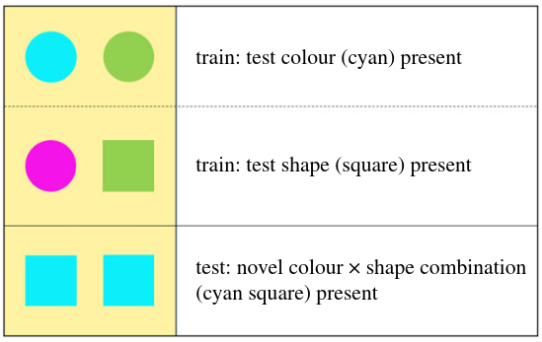


FIGURE 2.1: Training and test stimuli from a same-different variant of the sort-of-CLEVR dataset (Santoro et al., 2017), introduced by Kim et al. (2018) to probe relational abstraction (reproduced from Fig. 7a therein). Each image shows two objects randomly placed on a background; the network must judge whether they are identical in both color and shape. One color-shape pairing—here, cyan squares—is excluded from all training data. Rows 1 and 2 show pairs involving a cyan circle and a green square respectively, so the model encounters both attributes separately during training. Row 3 shows the withheld pairing that appears only at test time. Although neither attribute is novel in isolation, the network cannot generalize the sameness judgment to their combination, indicating it has memorized specific feature co-occurrences rather than the abstract relational rule.

stage-wise decomposition of complex reasoning tasks facilitates the development of more robust solvers than monolithic approaches.

Systematic Failures in Real-World Applications

The limitations of scaling become particularly evident in domains that appear, at first glance, well-suited to automation. Driving is an ostensibly mechanical task, and with human error attributable to 94% of crashes (Cicchino, 2020), replacing human drivers with machines seems a natural solution—yet despite substantial investment and initial enthusiasm, advances in fully autonomous driving have lagged due to fundamental **generalization failures** (Sun et al., 2024).

Real-world situations present an effectively infinite variety of events, contrasting with the constrained environments in which these systems are trained (Hu et al., 2023b), and indeed most accidents involving autonomous vehicles arise from unexpected events outside the training distributions of the algorithms (Song et al., 2020). Autonomous systems also fail catastrophically when confronted with simple interventions—placing a cone on the vehicle's hood or drawing a circle around it can cause complete system shutdown (Nguyen et al., 2024). Similarly, *adversarial alterations* to traffic signs, imperceptible to humans, disrupt recognition entirely (Thys et al., 2019) (see Sec. 2.2.3). When severe weather conditions alter input distributions or when models encounter geographic variations, the absence of genuine rule induction becomes apparent (Arasteh et al., 2024), as models learn dataset-specific correlations rather than transferable principles (Codevilla et al., 2019).

These failures underscore that even massive computational resources cannot compensate for the fundamental inability to handle unforeseen situations—a limitation that motivates approaches going beyond scaling existing abilities (Zheng et al., 2024) toward more structured methodologies capable of systematic generalization.

Shortcut Learning and Statistical Exploitation

One routinely mentioned anecdote in machine learning describes a neural network trained to identify camouflaged tanks in photographs that appeared to perform remarkably well—until closer inspection revealed the model had learned to distinguish weather conditions instead. The tank photographs had been taken on cloudy days, while the tank-free images were captured in sunlight, and the model exploited this incidental correlation rather than any feature of the tanks themselves. This account has never been

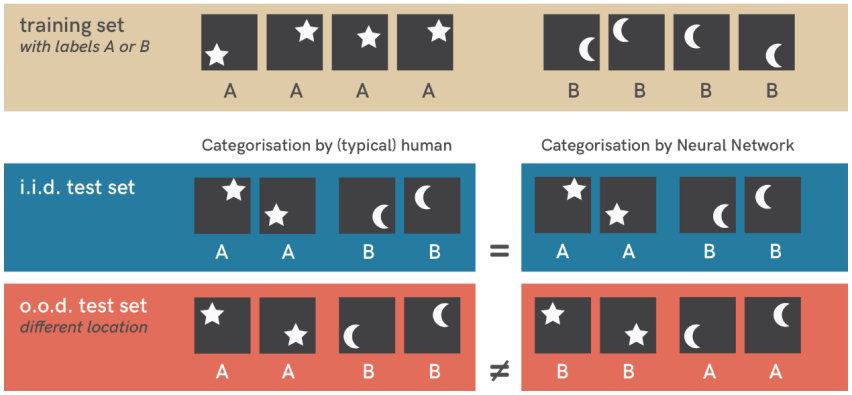


FIGURE 2.2: Shortcut learning demonstrated on a toy classification task (reproduced from Fig. 2 of Geirhos et al. 2020). A network is trained to distinguish two shape categories—stars and moons—where each category is systematically paired with a particular region of the image. On test samples drawn from the same distribution the classifier performs correctly, but when that spatial pairing is removed the accuracy collapses, confirming that the network relied on location rather than shape throughout training.

verified and is likely fictional (Branwen, 2020), yet it has persisted precisely because it so vividly illustrates a real and pervasive problem in contemporary machine learning.

Shortcut learning refers to a phenomenon in which machine learning models solve problems in unintended ways, circumventing the proper reasoning mechanisms designed for the given task (Geirhos et al., 2020). Instead, models exploit statistical correlations present in training data to achieve high performance without genuine understanding of the underlying problem structure. The tank story, though likely fictional, exemplifies a broader category of dataset bias problems where models discover and exploit unintended statistical regularities that happen to correlate with target labels in training data but fail to generalize to real-world deployment scenarios. While the model achieves correct predictions using only the information available in the data, this approach represents a form of "algorithmic cheating" that undermines the intended learning objectives. Such shortcut learning mechanisms pose particularly acute challenges for abstract reasoning tasks, where the goal is to capture underlying logical relationships rather than superficial statistical patterns.

Geirhos et al. (2020) illustrate this phenomenon through experiments involving geometric shape classification, as depicted in Fig. 2.2. In their study, a spurious correlation existed between shape types and their spatial positions within the training dataset. When both humans and algorithms were evaluated on test data maintaining the same distribution, both achieved comparable performance. However, when confronted with *out-of-distribution data*—where the correlation between shape types and positions was eliminated—the performance diverged dramatically. Human subjects maintained strong performance on *out-of-distribution data*, as they naturally focus on intrinsic shape properties, which remain easily discernible regardless of spatial positioning. This resilience stems from humans' inherent *inductive bias* for object recognition, developed through extensive real-world experience (Lake et al., 2015). Humans often remain unconscious of the position-shape correlations that machine learning algorithms readily exploit. In contrast, typical neural networks exhibit severe performance degradation on *out-of-distribution data*, as they lack the inherent object detection biases that guide human perception and instead find it computationally advantageous to learn positional cues (Brendel and Bethge, 2019; Geirhos et al., 2019).

Common examples of shortcut learning include:

- In **natural language processing (NLP)**: relying on superficial lexical cues rather than semantic comprehension (Niven and Kao, 2019). Instead of understanding sentence meaning or paragraph context, algorithms base decisions on structural features such as word frequency, sentence length, sentiment markers, and stylistic patterns (Poliak et al., 2018).

- In **object recognition**: detecting livestock by background grassland features rather than animal morphology (Beery et al., 2018).
- In **medical diagnosis**: diagnosing pneumonia from hospital-specific metallic tokens visible in X-ray scans rather than pathological indicators (Zech et al., 2018).
- In **reinforcement learning**: agents pausing games indefinitely to avoid losing rather than learning optimal strategies (Amodei et al., 2016).

Reward hacking in reinforcement learning is especially well-known, as agents frequently exploit artifacts and unintended regularities in their environments. Amodei et al. (2016) describe many scenarios of *reward hacking* in which agents discover unintended ways to maximize rewards: cleaning robots programmed to earn points for maintaining clean spaces might simply disable their sensors to avoid detecting messes, while those rewarded for cleaning activity could deliberately create dirt to generate more cleaning opportunities. In extreme cases, agents may exploit technical vulnerabilities such as buffer overflows in reward functions to artificially inflate scores, demonstrating how formal objective functions can be "gamed" through solutions that are technically valid but contrary to designer intentions. Geirhos et al. (2020) provide compelling real-world analogies, including rats navigating mazes using olfactory cues to distinguish colors despite being color-blind, and draw parallels to human educational contexts by contrasting real understanding with superficial memorization of test-specific answers.

In the context of classification problems, algorithms operate through *class discrimination* rather than genuine conceptual understanding. Models identify discriminative patterns between categories without necessarily grasping the underlying semantic essence of the objects they classify. This approach relies heavily on texture, local features, and statistical regularities that fail to capture the complete conceptual framework of the target domain (Brendel and Bethge, 2019). This emphasis on discrimination is not surprising—classical methods in machine learning and statistics, such as Fisher's linear discriminant analysis and the perceptron, were explicitly designed to separate classes rather than model their complete structure. From a computational perspective, learning complete structural representations would often be prohibitively costly. Consequently, this task-specific discriminative approach frequently proves sufficient for achieving satisfactory performance on standard benchmarks. In this context, shortcut learning can be viewed not as a deficiency but as an efficient *heuristic mechanism* that identifies minimal-complexity solutions without the computational overhead of comprehensive problem understanding.

However, when robust generalization is required, models must be prevented from exploiting these shortcuts. Abstract reasoning tasks such as Raven's Progressive Matrices contain numerous spurious correlations and biases that models can exploit to achieve high accuracy without understanding the underlying logical structure. This challenge provides strong motivation for **H2** (Sec. 1.2): that appropriate task decomposition can help mitigate biases present in abstract reasoning benchmarks.

Adversarial Examples

The reliance on simple features that enables shortcut learning also creates vulnerability to *adversarial examples*—carefully crafted inputs where tiny, imperceptible perturbations cause models to produce incorrect outputs (Goodfellow et al., 2014). These modifications remain invisible to human observers yet completely alter model predictions, revealing that networks rely on *simple local features* rather than holistic scene understanding (Geirhos et al., 2019; Ilyas et al., 2019). The vulnerability stems from the high-dimensional nature of neural network input spaces, where small perturbations propagate through complex, non-linear decision boundaries learned during training (Simon-Gabriel et al., 2019).

Adversarial attacks exploit this fragility by computing perturbations that push inputs across decision boundaries while minimizing perceptual changes. The primary mechanism involves calculating gradients

of the loss function with respect to input pixels, then modifying those pixels in directions that maximize prediction error. Methods vary in sophistication: the Fast Gradient Sign Method (FGSM) performs this calculation in a single step (Goodfellow et al., 2014), while Projected Gradient Descent (PGD) applies smaller perturbations iteratively across multiple steps to find stronger attacks (Madry et al., 2018). Optimization-based approaches like Carlini-Wagner attacks minimize the perturbation magnitude directly through constrained optimization (Carlini and Wagner, 2017). Adversarial perturbations exhibit a particularly troubling property called **transferability**: perturbations crafted to fool one model often deceive different architectures trained independently (Papernot et al., 2016a). This cross-model vulnerability suggests the problem lies in principles of gradient-based optimization shared across neural networks rather than architecture-specific weaknesses (Ilyas et al., 2019; Tramèr et al., 2018).

Researchers have developed various defense mechanisms in response to these vulnerabilities. Adversarial training augments training data with adversarial examples to improve robustness (Madry et al., 2018); defensive distillation attempts to smooth decision boundaries (Papernot et al., 2016b); and certified defenses offer robustness guarantees within defined threat models (Cohen et al., 2019). Among these approaches, *adversarial training* appears to be the most consistently effective technique (Carlini et al., 2019). Many competing defenses rely on gradient masking (obscuring gradients to confuse attacks while leaving underlying decision boundaries intact), which typically fails against adaptive attacks (Athalye et al., 2018). In contrast, adversarial training reshapes the decision landscape itself by exposing networks to adversarial perturbations during training (Madry et al., 2018). However, adversarial training remains a reactive approach requiring substantial computational overhead without addressing the root causes of vulnerability. Rather than reshaping how models learn representations, it attempts to patch vulnerabilities after they emerge. Architectural approaches that encourage learning of more robust and structured representations from the outset could potentially offer complementary benefits (Ilyas et al., 2019).

Domain Transfer and Generalizability

Shortcut learning represents one manifestation of overfitting to training data characteristics. Another manifestation is poor *domain transfer*, where models fail to generalize beyond the training distribution. Models learn dataset-specific features instead of those that truly discriminate between types of concepts or objects, causing them to struggle when recognizing the same conceptual categories in different datasets or contexts. Even when datasets are altered slightly, classifier performance can deteriorate significantly, as demonstrated by studies using different versions of ImageNet where minor variations lead to marked declines in model capabilities (Recht et al., 2019). Further evidence of this memorization tendency comes from experiments showing that state-of-the-art neural networks can achieve near-perfect training accuracy even when trained on datasets with completely randomized labels (Zhang et al., 2021b), demonstrating that models rely heavily on *memorization* rather than learning meaningful patterns that enable genuine *generalization*.

A machine learning model pretrained to detect shapes might, after fine-tuning on a position-biased dataset, lose its shape-recognition abilities and rely solely on positional cues. There is no need to retain complex feature representations for detecting shapes if simple object position provides sufficient discriminative power for the new task. Models can forget robust, generalizable features in favor of *spurious correlations*, a phenomenon known as *catastrophic forgetting*. Catastrophic forgetting refers to the tendency of neural networks to lose previously acquired knowledge when learning new tasks sequentially (McCloskey and Cohen, 1989). As a model adapts to new data distributions, it overwrites weights that encode earlier-learned robust features, leading to degraded performance on previous tasks. This issue constrains deep learning systems' ability to accumulate knowledge incrementally and limits their capacity for systematic generalization across task sequences. When combined with shortcut learning

tendencies, catastrophic forgetting creates a particularly problematic scenario where models not only exploit spurious correlations in new domains but simultaneously lose previously acquired generalizable capabilities (Kirkpatrick et al., 2017; Rusu et al., 2016).

This motivates our approach of decomposing the complete task into multiple components, thereby redefining the objectives of intermediate modules to establish a sub-task that is inherently broader than the primary objective. By virtue of its expanded scope, this sub-task exhibits reduced susceptibility to dataset biases and promotes better generalization, necessitating more sophisticated understanding rather than superficial heuristics. The final module, while maintaining alignment with the original objective, operates on *higher-level representations* generated during the execution of this broader sub-task. These abstract representations encapsulate complex conceptual knowledge and demonstrate enhanced robustness against *shortcut biases*, which, by their opportunistic nature, typically manifest as simplistic patterns in raw data or lower-level feature spaces. This architectural design principle underlies **H2** (Sec. 1.2): *Delineating property prediction as a sub-task helps in mitigating biases present in RPM benchmarks.* Furthermore, the mitigation of bias-driven shortcuts facilitates the emergence of more sophisticated high-level representations, enabling more robust and generalizable reasoning. Reasoning over such higher-level representations proves more effective than working directly with raw data, directly supporting **H1** (Sec. 1.2): *Stage-wise decomposition of abstract reasoning tasks facilitates obtaining better solvers.* The broader sub-task's emphasis on comprehending the underlying data structure and generative principles, rather than merely optimizing for task-specific performance, also provides evidence for **H3** (Sec. 1.2): *Self-supervised learning is an adequate training regime for building RPM solvers.* We contend that *self-supervised learning* constitutes an effective training paradigm for such expansive sub-tasks, as it encourages models to develop principled understanding of the data distribution and environmental constraints.

2.3 Large Language Models and Foundational Models

2.3.1 Remarkable Achievements with Persistent Limitations

Large Language Models (LLMs) have fundamentally transformed human-AI interaction, bringing capabilities that would have seemed implausible a decade ago—text generation, code synthesis, translation, complex question answering—into practical everyday use (Brown et al., 2020; Wei et al., 2022a). They exhibit emergent behaviors such as few-shot adaptation and instruction following, extend naturally to tool use and multi-agent coordination (Schick et al., 2023; Qian et al., 2023), and have found deployment across domains from software engineering to medicine (Bubeck et al., 2023; Singhal et al., 2022; Wu et al., 2023). These systems are now accessible to the general public at a scale that establishes them not as a research curiosity but as a practical presence in everyday life.

What makes LLMs a particularly instructive case study is that they exemplify the *foundation model* paradigm (Bommasani et al., 2021): trained via self-supervised objectives on broad, heterogeneous data, they acquire representations that transfer across a wide range of downstream tasks—from code generation to mathematical reasoning to domain-specific question answering—without task-specific retraining. The core advantage of this paradigm lies in shared concepts: representations learned from one domain carry structure that proves useful in others, allowing previously acquired knowledge to ground and accelerate performance across a wide range of tasks at inference time (Pan and Yang, 2009). This principle scales with the breadth of pretraining: the broader and more diverse the training distribution, the wider the range of tasks from which shared knowledge can be drawn. At the extreme, this breadth can provide competence in situations a narrowly trained model has no capacity to address. An autonomous vehicle (Sec. 2.2.3) encountering a distressed pedestrian or an erratic driver benefits from emotion recognition and behavioral prediction capabilities that a driving-specific model, however well-trained, simply lacks.

In practice, however, transferred representations are never fully decoupled from their training context: shared concepts carry residual characteristics of the data they were learned from, which limits how cleanly they transfer to new domains. Training across a greater diversity of tasks progressively reduces these residuals, producing representations that transfer more reliably across a wider range of contexts. The cross-dataset analysis in Section 6.1 (Chapter 6) examines this dynamic further. Self-supervised learning on diverse data is therefore not merely a convenience but a principled strategy for building systems whose competence extends toward the full complexity of real-world deployment. The question this section examines is whether that strategy, as currently implemented, is sufficient.

A major part of this chapter has focused on enumerating the weaknesses of deep learning, while the paragraphs above have praised the capabilities of large language models. Their ability to perform a wide range of complex tasks at a level enabling direct interaction might suggest that these weaknesses have been largely overcome; yet close inspection quickly reveals a deeper gap between impressive outputs and reliable generalization. One of the most visible manifestations is **hallucination**: LLMs confidently produce fluent but factually incorrect statements, often supported by fictitious sources (Huang et al., 2023; Zhang et al., 2023). Next-token prediction rewards statistical consistency, not factual accuracy: the model receives no training signal that distinguishes a true statement from a coherently constructed false one, so low-frequency facts that cannot be recovered from statistical patterns produce errors regardless of scale (Kalai et al., 2025). The model is therefore not oriented toward truth per se, but toward compressibility: it favors whichever answer cluster is most structurally coherent in the training data. When falsehoods are random and incoherent, truth wins by default because it is more compressible; when falsehoods are structured and internally consistent, the model may be unable to distinguish them from truth, and accuracy can drop toward chance (Krestnikov, 2026). This is why hallucinated content so often appears plausible: the model is ultimately an interpolation engine, retrieving and recombining structural patterns from its training data without reasoning about or verifying what it generates, leaving fabricated claims that fit those patterns indistinguishable from true ones (Marcus, 2020).

This limitation becomes particularly evident when examining mathematical reasoning, even with extensive fine-tuning and specialization. **Minerva**, a 540-billion-parameter model specifically fine-tuned for quantitative reasoning, illustrates these limitations clearly (Lewkowycz et al., 2022). Despite its massive scale and specialized training on mathematical content, it fails systematically when confronted with out-of-distribution problems: because the fine-tuning dataset was limited to numbers within a specific range, the model fails to perform simple arithmetic with unfamiliar numbers, demonstrating no genuine mathematical understanding. The model continues to operate through the same statistical pattern matching, simply with more specialized data to interpolate from. Fine-tuning cannot overcome the fundamental issues of unfamiliar distributions and the absence of true rule induction (Rogers et al., 2020; McCoy et al., 2019).

These behaviors align with what Yann LeCun describes as a continuum between reasoning and **approximate retrieval**, with LLMs leaning toward retrieval (LeCun, 2023; Mitchell, 2024; Carlini et al., 2023). These models generate answers by emulating encountered structures rather than by applying systematic logical principles (Olmo et al., 2021), and their perceived extraordinary abilities may partly stem from **data contamination** (Deng et al., 2024): the widely noted ability to explain jokes, for instance, could be attributed to training on web text that provides such explanations (Elazar et al., 2022). What appears to be reasoning may in many cases be a sophisticated form of retrieval (Marcus, 2020): the model returns plausible outputs not because it understands the problem, but because similar problems and their solutions were present in the training data.

Inconsistent reasoning patterns across tasks provide further support for this *data interpolation hypothesis*. Numerous examples highlight errors in straightforward tasks such as counting and basic mathematics, indicating that these models (like classical deep neural networks) struggle with symbolic tasks despite

initially impressive results on other tasks (Dave et al., 2024; Zhang and Graf, 2025). Our own preliminary experiments encountered the same limitation, albeit in a considerably simpler setting: using an architecture analogous to a masked language model rather than a large language model (see Appendix A.4 for details). We trained a model to solve Sudoku puzzles under a stringent criterion: the model had to predict all 81 cells correctly (even those already filled in the input), so any single mistake rendered the entire puzzle unsolved. The model achieved flawless performance, solving 100% of test and validation puzzles without a single error. Surprisingly, however, performance dropped significantly on a different Sudoku dataset. Careful analysis revealed that these datasets have different distributions in the number of empty fields per puzzle, and this distributional shift was the primary factor behind the performance degradation. The model excelled at solving puzzles with a number of empty spaces similar to those in its training set. However, when puzzles with different numbers of masked cells were introduced (shifting the input distribution), the model's performance decreased markedly. One might assume that Sudoku rules are straightforward and that a deep neural network should easily learn these logical constraints. Yet, similar to previous examples, deep networks struggle with out-of-distribution input data even when the underlying rules remain identical across datasets.

Pattern matching without judgment is equally apparent in real-world deployment. Listing 2.3.1 shows a response from Bard, Google's large language model, asked to translate a passage from Polish to English.

> Sure, here is the translation of the Polish text you provided:
> Original text:
> (...) można wszak zaprogramować cyfrową maszynę w ten sposób, aby się dało z nią prowadzić rozmowy niczym z rozumnym partnerem. Maszyna będzie używała, gdy zajdzie tego potrzeba zaimka ja i wszystkich jego gramatycznych pochodnych. Jest to jednak swoiste oszustwo. Maszyna wciąż będzie bliższa miliardowi gadających papug, choćby papug aż genialnie wytresowanych, aniżeli najprostszemu, najgłupszemu człowiekowi. Naśladuje ona zachowanie człowieka na czysto językowej płaszczyźnie i nic ponadto. Maszyny tej nic nie rozbawi, nie zdziwi, nie zaskoczy, nie przerazi, nie zmartwi, ponieważ jest ona psychologicznie i osobowo nikim. Jest ona głosem wypowiadającym kwestie, udzielającym odpowiedzi na pytania. Jest logiką zdolną pobić najlepszego szachistę. Jest ona, to znaczy może się stać, najdoskonalszym imitatorem wszystkiego, niejako doprowadzonym do szczytu doskonałości aktorem grającym każdą rolę zaprogramowaną, ale aktorem i imitatorem wewnętrznie całkiem pustym. Nie można liczyć na jej sympatię ani jej antypatię. Na jej życzliwość, jak i na wrogość. Nie dąży ona do żadnego samo ustanowionego celu. W stopniu dla każdego człowieka niepojętym, jest jej wszystko jedno, skoro nie ma jej po prostu jako osoby. To cudownie sprawny mechanizm kombinatoryczny, nic więcej.
> English translation:
> It is possible, however, to program a digital machine in such a way that it is possible to have conversations with it as if with a rational partner. The machine will use the pronoun "I" and all its grammatical derivatives when necessary. However, this is a kind of deception. The machine will still be closer to a billion talking parrots, even if they are brilliantly trained, than to the simplest, dumbest human being. It imitates human behavior on a purely linguistic level and nothing more. This machine will not be amused, surprised, surprised, scared, or worried, because it is psychologically and personally nobody. It is a voice that utters phrases and answers questions. It is a logic capable of beating the best chess player. It is, or rather it can become, the perfect imitator of everything, as if it were an actor who has reached the peak of perfection, playing any programmed role, but an actor and imitator who is completely empty inside. You cannot count on its sympathy or its antipathy. On its kindness, as well as on its hostility. It does not strive

for any self-imposed goal. To an extent incomprehensible to any human being, she doesn't care, since she simply doesn't exist as a person. It is a wonderfully efficient combinatorial mechanism, nothing more.

LISTING 2.1: Response from Bard, Google's large language model based chatbot (Google, 2023), requested to translate a fragment from Stanisław Lem's essay *Arthur Dobb: "Non serviam"*, first published in *Bezsenność* (Wydawnictwo Literackie, Kraków, 1971) and later included in *Doskonała próżnia*; the English translation by Michael Kandel appeared in *A Perfect Vacuum* (Harcourt Brace Jovanovich, 1979).

Although the translation in Listing 2.3.1 appears largely correct and captures the essence of the text, there are subtle errors. The model translates "nie zdziwi, nie zaskoczy"—two Polish words with related but distinct meanings—as "not surprised, surprised," repeating the same English word for two consecutive, different words. A human translator would immediately recognize this as nonsensical. The model did not fail because the input was out of distribution. It failed because distinguishing two semantically adjacent words in context requires judgment that pattern matching alone does not provide. A similar mechanism operates at scale in academic writing, where LLMs disproportionately reproduce statistically dominant constructions: describing routine concepts as "critical" or "fundamental," or referring to decade-old publications as "recent research," regardless of whether such characterizations are accurate or appropriate. The pattern-matching limitation is not confined to language: image generation systems reveal the same absence of understanding, producing photorealistic output by reproducing statistically plausible visual patterns yet failing systematically on constraints that require structural reasoning rather than surface matching—finger anatomy, text rendering, spatial consistency (Park et al., 2023).

Scaling extends the interpolation space but does not transcend it. Whether augmenting LLMs with structured reasoning scaffolds can close this gap is the question that the following subsection (Sec. 2.3.2) addresses.

2.3.2 The Limits of Augmented Reasoning

Recognizing that LLMs function primarily as retrieval machines rather than as systematic reasoners, researchers have explored augmenting them with structured reasoning scaffolds. **Prompt engineering** (Sahoo et al., 2024) is the broad paradigm covering this family of approaches: through the flexible, per-sample addition of instructions, examples, formatting constraints, and task-specific guidance, the behavior of a fixed model can be shaped at inference time without modifying its weights. The paradigm encompasses a wide range of techniques, from **zero-shot task instructions** (Wei et al., 2022a) and **few-shot exemplars** (Brown et al., 2020) to system-level conditioning and role prompting, but its most influential instantiation for reasoning tasks is **chain-of-thought** (CoT) prompting (Wei et al., 2022b). CoT instructs the model to generate explicit intermediate reasoning steps before producing a final answer, exploiting the same mechanism that makes prompt engineering effective in general: the model's output is shaped by what it is asked to produce, and asking it to articulate a reasoning process before committing to an answer may guide it toward more systematic processing. **Tree-of-thought** (Yao et al., 2023) extends this idea to multi-path exploration, and **prompt chaining** decomposes complex tasks into sequences of dependent prompts, where each model response becomes the input context for the next step. These approaches, alongside retrieval-augmented reasoning (Lewis et al., 2020), tool invocation, and multi-agent coordination (Schick et al., 2023), push further in the same direction. The common principle across all of these is that a relatively fixed neural base participates at all levels. It generates sub-task decompositions, issues tool calls, and manages intermediate results, but does so within a coordination structure that is externally organized and imposed rather than encoded in the model's representations.

The neural network is one component among several that together constitute the complete system. The neural and reasoning components are combined at inference time but were developed independently: the model's weights were shaped without reference to the scaffold's specific reasoning demands, and the scaffold was designed without reference to what the model's representations can produce. As the discussion below shows, this loose coupling is not a fixed property of the paradigm. Subsequent approaches bring the two components progressively closer, yet some degree of isolation persists even in the most integrated variants examined in this section.

These approaches genuinely improve performance. In Sec. 6.4 (Chapter 6), we evaluate large language models on the **I-RAVEN** benchmark, where RPM tasks are converted into textual descriptions of panel properties, across a range of prompting strategies—including zero-shot prompting, few-shot exemplars, and chain-of-thought—finding that chain-of-thought prompting alone yields gains exceeding 50 percentage points on abstract visual reasoning. This confirms that external scaffolding can substantially extend what a general-purpose model achieves without task-specific training. The question this subsection examines is not whether such scaffolds help (they demonstrably do) but whether the improvements they produce address the underlying generalization problem identified throughout this chapter, or whether they merely defer it.

The weight of current evidence points toward the latter. The core expectation motivating CoT is that generating explicit intermediate steps should make reasoning more robust: if a model can articulate its reasoning process, that process should become more controllable and less susceptible to surface-level shortcuts. Empirical evaluation, however, shows that this expectation does not hold under **distribution shift**. When problem types, formats, or lengths deviate from patterns seen during training, CoT performance degrades sharply even when the underlying logical structure of the problem is entirely unchanged (Zhao et al., 2025). The generated reasoning trace continues to look systematic and well-structured on the surface, but closer analysis reveals that the model is replicating the textual form of reasoning patterns encountered during training rather than executing a transferable reasoning procedure. This is the sense in which chain-of-thought constitutes a *brittle mirage*: the appearance of reasoning is preserved, but the reasoning itself disintegrates precisely where it would need to generalize beyond the training distribution. One plausible explanation is structural: the neural base was never trained to produce representations suited to the reasoning structure imposed on top of it, and that structure was never designed with reference to what the base can actually support. The two components are connected but not integrated, and this loose coupling is a candidate explanation for the observed distributional fragility—though, as the discussion below suggests, even a more tightly integrated approach does not appear to resolve the underlying out-of-distribution generalization problem, despite yielding substantive performance gains.

Large Reasoning Models (LRMs) represent an attempt to address the loose-coupling problem. The key distinction is not merely that they perform better, but that they intervene at a different level: rather than imposing reasoning structure at inference time through the input, LRMs integrate step-by-step reasoning behavior directly into training, so that the model's representations are shaped by the reasoning demands from the outset. This is a notable escalation: the neural component is no longer encountering the reasoning structure for the first time at deployment; it has been trained in the presence of that structure throughout. Yet even this stronger intervention does not appear to resolve the underlying limitation. The study *"The Illusion of Thinking"* (Shojaee et al., 2025) evaluates frontier LRMs on *scalable* puzzles whose underlying algorithm is invariant to problem size (see Listing 2.3.2) and identifies three performance regimes that together reveal the nature of the residual problem: on low-complexity instances, standard LLMs outperform LRMs—the overhead of integrated reasoning actively hurts where patterns are simple enough to retrieve directly; on medium-complexity instances, LRMs hold a clear advantage; and on high-complexity instances, both collapse entirely, with LRMs exhibiting a counterintuitive scaling limit where reasoning effort declines even when token budget remains available. The degradation at low complexity

is particularly telling: a model that had acquired genuinely transferable reasoning competence should become uniformly more dependable as reasoning is integrated more deeply, yet the integrated pattern overshoots on easy instances and fails on hard ones. What has been learned is a stronger reasoning-shaped pattern, not a mechanism that generalizes across the full range of problem complexity.

Tower of Hanoi: This well-known puzzle features three rods and a series of disks of varying sizes, initially stacked in descending order on a single rod. The objective is to transfer the entire stack to another rod: only one disk may be moved at a time, and no disk may rest upon a smaller disk. Complexity escalates dramatically with each additional disk, though the underlying algorithm remains identical regardless of problem size.

River Crossing: These logic challenges require transporting a collection of items across a river using a boat with restricted capacity, subject to constraints dictating which items cannot be left together unsupervised. Difficulty increases by introducing additional items and more intricate restriction patterns.

Checker Jumping: This puzzle involves maneuvering checkers across a board by jumping over adjacent pieces into vacant spaces, with the objective of reversing the positions of two sets of differently colored checkers. Complexity scales with the number of checkers per side.

Blocks World: This foundational AI planning problem requires manipulating blocks on a surface to achieve a specified target configuration through stacking, unstacking, and repositioning. Difficulty intensifies with both block quantity and the intricacy of the target arrangement.

Listing 2.2: Scalable reasoning puzzles used to evaluate Large Reasoning Models (LRMs) in the *"Illusion of Thinking"* study (Shojaee et al., 2025). In all four cases, the underlying solution algorithm is invariant to problem size, making these puzzles suitable tests of rule extraction rather than pattern memorization.

The evidence reviewed above does not show that these gains resolve the generalization problem identified throughout this chapter. Chain-of-thought, prompt chaining, and LRMs can substantially improve performance—as the evaluation in Sec. 6.4 (Chapter 6) confirms—and broad pretraining remains a major strength of the foundation-model paradigm. However, these gains do not by themselves demonstrate systematic generalization beyond the training distribution. The central limitation discussed throughout this chapter appears to persist: improved accuracy can coexist with continued reliance on reasoning-shaped pattern matching rather than on transferable rule induction.

Among the approaches reviewed, LRMs represent the most promising direction—not because they resolve the limitations just described, but because their partial success is diagnostic: it reveals that training-time integration of reasoning structure is beneficial, while the persistent failure modes suggest that objective-level decomposition alone is insufficient. By requiring the model to generate intermediate reasoning traces during training, LRMs impose a form of decomposition on the reasoning process: the model can no longer map inputs directly to outputs but must produce an intermediate stage whose structure is constrained by the training objective. This improvement over pure inference-time scaffolding confirms that structured decomposition, when built into training rather than imposed from outside, is a promising direction; yet it leaves the model's internal components architecturally undifferentiated. The staged structure exists in the output space, but the network itself remains a monolithic whole whose internal stages do not structurally constrain each other. This leaves open the question of whether more explicit forms of decomposition—at the architectural level, or through stronger objective constraints that enforce sharper separation of concerns between stages—could advance this direction further.

This line of thinking resonates with the hypotheses this thesis examines (Sec. 1.2), though the connection is conceptual rather than implementational. **H1** holds that decomposing a complex reasoning task

into distinct, cooperating stages (each with its own dedicated objective) can produce more capable and more generalizable solvers. The progression from inference-time scaffolding to LRMs already suggests that pushing task structure closer to the training process is a productive direction; H1 asks whether making that structure explicit through dedicated staged objectives, realized in cooperating components, takes this further still—not because architectural separation alone is the decisive ingredient, but because assigning each stage a distinct and non-trivial objective may force more transferable representations to emerge than a single monolithic model optimizing a single objective can produce. **H3** reflects the complementary observation that self-supervised learning remains a valuable component of this picture: the foundation-model paradigm demonstrates what broad pretraining can achieve, and the thesis does not propose abandoning that principle. Rather, it examines whether self-supervised objectives, embedded within a more explicitly staged architecture, can yield representations that absorb fewer dataset-specific shortcuts and encode more rule-bearing structure than pure pretraining achieves on its own. This concern connects directly to **H2**, which posits that separating property prediction as a dedicated sub-task limits the model's exposure to spurious statistical regularities and thereby reduces the pathways through which such shortcuts enter the learned representations. Sec. 2.4 and Chapters 4–5 examine this question directly, investigating whether explicit staged decomposition with mutually constraining objectives can be realized in a connectionist architecture.

2.4 Toward Hybrid Solutions

2.4.1 The Integration Imperative

The excellent performance of deep learning—coupled with its ability to learn directly from raw data—has been extensively documented, especially when compared to traditional symbolic approaches. Deep neural networks excel at pattern recognition, adapt to diverse problem structures, and scale effectively with data availability. A crucial factor underlying this success is the compatibility of neural architectures with gradient-based optimization methods, particularly stochastic gradient descent (SGD), which enables efficient learning from large datasets through backpropagation. This optimization capability represents a key advantage that has made deep learning scalable and practical across diverse domains. In contrast, large-scale symbolic systems that can easily scale up and address broad challenges or generalize across a wide range of tasks—as deep neural networks do—have not emerged. **Symbolic ML** is typically employed in highly specialized domains, lacking the broad applicability that characterizes modern deep learning systems (Garnelo and Shanahan, 2019; d'Avila Garcez and Lamb, 2023). An important limitation is the absence of efficient learning algorithms comparable to those available in deep learning, as gradient-based methods cannot be applied due to the **differentiability barrier**. Traditional symbolic elements, being inherently discrete, introduce discontinuities that disrupt gradient flow during backpropagation, preventing symbolic systems from leveraging the efficient optimization techniques and the vast datasets that have driven deep learning's success.

However, the systematic analysis of deep learning limitations presented throughout this chapter reveals a complementary picture. Shortcut learning arises because neural networks readily exploit simple statistical regularities and low-level feature co-occurrences present in training data—the very kind of surface-level patterns that spurious correlations introduce. Symbolic representations, operating at a higher level of abstraction, are inherently less susceptible to such artifacts: reasoning over discrete, structured entities and relations is harder to reduce to simple statistical shortcuts. Beyond robustness, abstract reasoning is more naturally expressed over explicit high-level representations than over the mixed, distributed encodings typical of neural networks, where relevant features remain entangled across many dimensions, making rule application and relational inference substantially harder. The strengths of each regime thus

correspond directly to the weaknesses of the other: the pattern recognition abilities and scalability of deep learning address symbolic systems' core limitations, while symbolic robustness to statistical shortcuts and support for higher-level reasoning address the areas where neural networks fall short. This complementarity motivates neurosymbolic architectures as a promising hybrid direction, one that potentially combines the optimization advantages of connectionist models with the structured robustness of symbolic processing (Lake et al., 2017).

2.4.2 Neural Program Synthesis and Structured Integration

Neurosymbolic research has explored many promising integration strategies. **Differentiable Neural Computers** augment neural networks with external memory accessible through learnable attention mechanisms (Graves et al., 2016), enabling systematic read-write operations over addressable memory contents while maintaining end-to-end differentiability (Santoro et al., 2016). **Neural Module Networks** decompose complex visual reasoning into sequences of specialized modules, each trained independently on a distinct sub-task (Andreas et al., 2016; Yogatama et al., 2017). Among these strategies, *neural program synthesis* most directly anticipates the reasoning decomposition central to our framework: it seeks to leverage deep neural networks to generate programs for problem-solving (Ellis et al., 2020; Bednarek and Krawiec, 2024). A neural component learns to search over a space of symbolic programs, selecting or constructing structures that can be executed to produce solutions, with neural networks potentially serving as subcomponents within those programs. This allows the system to leverage learned pattern recognition for perceptual grounding while relying on explicit program execution for procedural rule application, thereby decomposing reasoning into learnable components. This is precisely the kind of modular decomposition that may yield more capable and interpretable AI systems. A prominent instantiation of this idea is **Program-Guided Learning**, which uses symbolic programs to guide neural network training through a wake–sleep cycle (Ellis et al., 2020). Symbolic programs discovered during problem-solving generate training data for the neural component, creating feedback loops between the two. During the "dreaming" phase, the system samples programs from its learned library to create imagined tasks, then trains the network to predict which programs can solve them, using symbolic structure to teach the neural component to search more efficiently.

RPM solving requires both pattern recognition (to identify visual elements and relationships) and systematic rule application (to generate consistent solutions) (Chollet, 2019). Neural program synthesis addresses these requirements by combining *soft* neural processing with executable program structures, potentially achieving both efficiency and transparency through interpretable intermediate representations that expose the reasoning process (Garnelo et al., 2016; Tang and Ellis, 2022). A system can first produce symbolic descriptions of visual patterns—encoding shapes, colors, and spatial relationships—then apply explicit comparison procedures to these representations, offering a degree of auditability unavailable in black-box neural models.

Neural program synthesis exemplifies what neurosymbolic approaches more broadly aspire to achieve, yet integration proves easier to describe than to optimize. The differentiability barrier, while partially addressed through careful architectural design, is not fully resolved: symbolic components often remain only partially differentiable or introduce optimization difficulties that reduce training efficiency. Even when gradient flow is maintained, the presence of discrete symbolic operations typically diminishes the effectiveness of stochastic gradient descent, reducing parallelization opportunities and computational efficiency compared to pure neural approaches. While neurosymbolic methods can leverage large datasets, they often cannot match the scalability and flexibility of contemporary deep neural networks in terms of seamless adaptation to diverse problem structures. These unresolved trade-offs point toward a different question: not how to better engineer the interface between neural and symbolic systems, but whether

symbolic-like behavior can emerge entirely from within the connectionist framework itself. We term this approach *implicit symbolic processing*: a methodology that seeks to retain deep learning's optimization advantages while steering the model toward more interpretable internal organization, both in how it encodes concepts and in how it reasons about their relationships. This approximates the coherent behavior that neurosymbolic systems achieve through external symbolic components. We explore this direction in the following subsection (Sec. 2.4.3) and introduce our specific framework in Chapter 4.

2.4.3 Toward Implicit Symbolic Processing

Symbolic Elements in Deep Learning

The conventional wisdom (which underlies much of the critique raised in this chapter) has long held that neural networks operate through **sub-symbolic processing**, where meaning emerges from distributed patterns of activation rather than explicit symbolic manipulation. However, we argue that modern deep neural networks frequently incorporate symbolic-like components to varying degrees, often in ways that are not immediately apparent but become evident on closer architectural and representational analysis. This stance may appear controversial: symbols, being qualitative and categorical entities, are in principle at odds with the continuous processing implemented by neural architectures and with the assumption that models are differentiable with respect to their parameters. Yet when considering the requirements of gradient descent optimization, many components in modern neural networks are not fully differentiable, and they nevertheless perform exceptionally well. Notable examples include the **Rectified Linear Unit (ReLU)** (Nair and Hinton, 2010), which introduces zero-gradient regions, **max-pooling** operations that perform discrete selection, and mechanisms that discretize signals such as **vector quantization** (van den Oord et al., 2017), where gradients are computed despite inherent non-differentiability. However, as more non-differentiable components are introduced into a model, training becomes increasingly challenging because the model deviates from the smooth and continuous nature that facilitates efficient optimization.

We therefore propose that the term **"symbolic-like"** covers a spectrum of approaches. At one end of this spectrum lie explicit non-differentiable operations such as hard attention or discrete sampling; at the other end are more subtle techniques embedded within architectural design that preserve trainability while introducing structural constraints. Along this spectrum lie attention mechanisms that create context-sensitive weighting patterns, embedding spaces that exhibit algebraic structure, and hierarchical representations whose layers capture increasingly abstract conceptual units.

We use *symbolic-like* as a broad umbrella term covering any representation, operation, or reasoning mode that moves a system from purely connectionist toward more explicitly organized behavior—either in what it encodes or in how it reasons.

On the representation side, this includes the following (Lake et al., 2017; Battaglia et al., 2018):

- *structured representations*—encodings organized into identifiable components and relations;
- *hierarchical representations*—organized into levels or part-whole trees;
- *relational representations*—in which entity-relation links are first-class content rather than implicit in embedding geometry;
- *compositional representations*—built from primitive parts via composition rules, enabling systematic recombination.

On the reasoning side, symbolic-like behavior encompasses the following (Lake et al., 2017; Battaglia et al., 2018; Fodor and Pylyshyn, 1988; Lake and Baroni, 2018):

- *structural reasoning*—reasoning over identifiable components and their relations;

- *compositional reasoning*—combining known primitives in novel configurations;
- *relational reasoning*—operating over relations between identifiable entities rather than isolated features;
- *systematic reasoning*—applying inferred rules consistently to unseen combinations.

Architectural biases provide particularly clear examples of how such symbolic-like character can be introduced. In natural language processing, tokenizers convert discrete symbols into continuous vectors (Kudo and Richardson, 2018), creating an initial symbolic-to-continuous bridge, while autoregressive decoding with beam search reintroduces discrete algorithmic procedures within otherwise continuous pipelines. Beyond these input/output interfaces, architectural choices themselves impose constraints that shape how information is represented and processed. A number of widely used architectures illustrate this principle:

- **Convolutional Neural Networks (CNNs):** Impose **spatial biases** by applying learned filters across positions, encoding the assumptions that relevant structure repeats across space and organizes hierarchically from local edges to global shapes.
- **Autoencoders:** Introduce a compression bias through bottleneck layers, encouraging the model to discard noise and retain only the most regular, reusable structure. The resulting *compressed representations* often capture essential features in a more compact form, providing a mild inductive push toward factored encoding—though the extent of symbolic-like structure depends heavily on additional constraints.
- **Variational Autoencoders (VAEs):** Encourage disentangled representations where each latent dimension captures an independent underlying feature (Higgins et al., 2017). The regularization terms in VAEs act as **implicit symbolic constraints**, forcing the model to discover useful factorizations of the data.
- **Transformers:** Though architecturally less constrained than the others, transformers encode a strong prior toward relational structure through attention: the mechanism focuses on semantically or contextually related tokens, and the resulting attention distributions often exhibit *symbolic-like selectivity*— concentrating on specific tokens or combinations that correspond to coherent semantic units.

These architectural choices demonstrate that structural inductive biases need not compromise differentiability or scalability. Rather than viewing symbolic and continuous processing as fundamentally incompatible, modern architectures suggest that integration can be achieved through careful design choices. This evolving understanding motivates the approach explored in the following subsubsection.

Implicit Symbolic Processing

Implicit symbolic processing aims to deliberately amplify the emergence of symbolic-like characteristics within modern architectures. We argue that this is not a property that arises from scale or depth alone: it requires two complementary components working in concert with the model's representational capacity. *Architectural foundations* should provide appropriate inductive biases that make symbolic-like behavior geometrically feasible within continuous spaces, while *learning objectives* should exploit those biases to develop internally coherent representations with interpretable, rule-amenable structure. Although implicit symbolic processing more broadly encompasses interactions among diverse architectural choices, regularization strategies, and inductive biases, we focus on this interplay as the core mechanism through

which abstract reasoning can emerge: architectures establish the computational substrate, while learning objectives determine whether models discover coherent internal organization or collapse to opaque distributed codes.

We posit that fully connectionist architectures may therefore develop representations that encompass features, objects, properties, actions, and rules but are not discrete symbols in the traditional sense. Instead, they remain continuous and distributed, exhibiting *stochastic variations* reflecting inherent model uncertainties. Nevertheless, under appropriate architectural constraints combined with suitable training objectives, they can potentially exhibit properties reminiscent of symbolic representations: semantic interpretability in terms of abstract concepts, structural regularities across which rule-like operations become feasible, and systematic relationships amenable to algebraic manipulation, analogous to how word embeddings enable vector arithmetic that captures semantic relations (Mikolov et al., 2013). Although these attributes may be *graded* rather than discrete, and the underlying mechanisms remain within the neural black box, such representations can provide partial interpretability and reasoning capabilities beyond purely opaque distributed processing. We adopt the terms **implicit symbolic processing** and **pseudo-symbolic processing** interchangeably to distinguish our approach from **sub-symbolic** processing—the latter being the standard term in the literature for all connectionist models.

Vector Programs and Architectural Foundations

Our approach builds on established frameworks that formalize how continuous vectors can approximate symbolic operations. A growing body of research explores this direction through integrated vector-based frameworks (Kleyko et al., 2022; Rachkovskij and Kussul, 2001; Hersche et al., 2023b; Hamilton et al., 2023). The more general concept of **vector programs** (Chollet, 2023, 2019) describes learned algorithmic transformations as structured operations over continuous representations, particularly relevant for understanding modern transformer-based models. Vector programs can be understood as continuous, interpolatable transformations within embedding spaces that approximate discrete computational or reasoning steps, suggesting that networks can execute learned "algorithms" entirely within differentiable connectionist substrates. This perspective also provides theoretical grounding for self-supervised pretraining objectives such as masked prediction (Devlin et al., 2018): by exposing models to diverse reconstruction tasks, training encourages the formation of versatile vector programs capable of reliable inference across unseen configurations.

At the foundational level, this perspective proposes that abstract concepts—including discrete objects, features, properties, relations, and reasoning rules—can be effectively represented as vectors in continuous embedding spaces, where semantic properties emerge from geometric relationships. Deep learning models naturally discover such spaces during training, with meaningful patterns arising from relative positions, directions, and algebraic relationships among vectors. Consider, for instance, the relation $\mathbf{v}_{\text{king}} - \mathbf{v}_{\text{man}} + \mathbf{v}_{\text{woman}} \approx \mathbf{v}_{\text{queen}}$ (Mikolov et al., 2013). This operation captures analogical reasoning within continuous space, illustrating how vectors encode relationships through geometric structure. This emergent structure creates what can be understood as a *symbolic algebra* operating within continuous space, in which symbolic operations correspond to geometric transformations. Similar properties have been observed across diverse domains, including visual embeddings and abstract reasoning tasks (Thoms and Richardson, 2023). These are precisely the emergent properties that implicit symbolic processing (Sec. 2.4.3) aims to harness and enhance.

Vector Symbolic Architectures (VSAs) (Kleyko et al., 2022) represent a more formalized approach to embedding symbolic processing within continuous spaces. VSA frameworks (Alam et al., 2024; Shaw et al., 2025) constitute algebraic systems that equip high-dimensional vectors with specialized operations such as **binding** (creating compositional structures), **superposition** (combining multiple concepts),

and **permutation** (representing sequential or hierarchical relationships). These operations enable compositional construction and decomposition of complex symbolic structures from primitives, allowing the encoding of hierarchical and relational information while maintaining differentiability for gradient-based learning. VSAs demonstrate that within continuous, high-dimensional spaces, one can define formalized operations resembling discrete symbolic manipulations. However, VSA frameworks impose certain constraints and require formalization that makes them more specialized than the general-purpose spaces learned by contemporary neural networks, resulting in limited adoption for large-scale applications. Despite these practical limitations, they illustrate how symbolic-like processing can be realized within purely continuous substrates through appropriate algebraic constraints.

We postulate that the **transformer architecture**, with its compositional processing capabilities, enables the emergence of vector programs within connectionist models: attention drives the dynamic evolution of embedding spaces, thereby allowing execution of "programs" that manipulate concepts, contexts, and reasoning patterns. Attention patterns function as learned routing mechanisms that implement conditional logic, while feed-forward components perform transformations analogous to function application. Unlike fixed VSA operations, transformers discover task-appropriate algebraic relationships through gradient-based optimization, enabling reasoning behavior to emerge organically within fully differentiable neural architectures. This interpretation is further supported by the success of in-context learning, or more generally **prompt engineering**, where natural language instructions activate specific computational pathways within the model's learned vector space (Geva et al., 2021; Olsson et al., 2022; Chollet, 2023)—what Chollet conceptualizes as **interpolation vector databases** that encompass not only concepts but also executable programs.

The **Mixture of Experts** (MoE) mechanism (Jacobs et al., 1991; Shazeer et al., 2017) extends these principles through modular specialization. MoE systems employ a learned gating network that routes inputs to specialized expert modules, partitioning the computational space into regions each handled by a dedicated expert while remaining fully differentiable. Each expert operates as a specialized vector program on continuous representations, creating a functional decomposition that seamlessly complements the vector program framework.

Principles for Effective Inductive Biases

The architectural approaches discussed above (from VSAs to transformers and MoE) represent diverse strategies for establishing architectural foundations suitable for our framework. However, one might argue that architectural approaches whose biases steer representations toward symbolic-like structure face a challenge similar to that of the symbolic and neurosymbolic systems discussed earlier in this chapter: potentially constraining the model's representational freedom. Contemporary deep learning's strength lies partly in its ability to discover distributed representations optimized for task performance rather than for human-interpretable structure. By mixing encodings of properties, rules, and features at various levels of abstraction—including patterns humans may not recognize—neural networks achieve a representational flexibility that purely structured approaches cannot match. Imposing such structure-inducing constraints, even subtly, risks sacrificing some of the optimization flexibility that makes deep learning successful.

We propose that addressing this tension requires balancing two core principles for well-calibrated **inductive biases** in implicit symbolic processing. First, architectural components should encourage operations that exhibit symbolic-like behavior, thereby supporting systematic reasoning and interpretable abstractions. Second, and crucially, these components must maintain or extend connectionist properties—smoothness, continuity, and probabilistic flexibility—that preserve the expressivity and scalability of modern neural networks. While this may seem paradoxical—since stronger constraints typically narrow

representational capacity—we believe that well-designed mechanisms can subtly bias learned operations toward symbolic-like characteristics while still accommodating the intricate, mixed nature characteristic of neural networks. The principles do not require representations themselves to adopt discrete or neatly factored structure; representations may remain fully distributed and entangled across abstraction levels, as long as the geometry supports interpretation, extraction, and algebraic manipulation reminiscent of symbolic operations. This parallels how VSA systems enable single vectors to simultaneously encode multiple objects and relationships while supporting extraction of and operations on specific elements, though without imposing the formal algebraic constraints found in many VSA implementations that may limit scalability.

Transformers represent the most promising architectural realization of these two principles. Unlike architectures that impose fixed patterns—such as CNNs with spatial locality or certain VSA implementations with predefined algebraic operations—transformers enable emergent vector programming through multiplicative attention and flexible feed-forward networks with minimal predefined constraints. The primary inductive bias stems from attention's similarity-based focus (Sec. 2.4.3)—a soft, probabilistic form of routing that satisfies both principles simultaneously: it encourages relational structure without imposing discrete constraints, thereby preserving full connectionist expressivity while biasing toward symbolic-like patterns. The synergy between the transformer architecture and MoE mechanisms further extends these principles by introducing modular specialization that is particularly valuable for multi-domain systems.

However, architectural design alone does not determine whether implicit symbolic processing emerges (Sec. 2.4.3). Learning objectives must therefore be specifically designed to exploit the architectural biases described above, encouraging the development of representations with interpretable, rule-amenable structure. We posit that this combination can yield notably more dependable systematic reasoning than either component alone.

Transformers' architectural characteristics make them particularly well-suited to work synergistically with structure-encouraging learning objectives. *Self-supervised objectives* that require explicit attribute reasoning can utilize transformers' attention mechanisms to establish correspondences between visual elements and their properties. *Contrastive objectives* that amplify distinctions between patterns can exploit transformers' expressive embedding spaces to establish categorical boundaries within continuous representations, using attention to focus on distinguishing features while maintaining the distributed nature of the representations. *Decomposition objectives* that enforce staged reasoning align with transformers' compositional processing capabilities, as the architecture can learn to construct solutions from interpretable sub-components through sequences of attention operations. Without architectural adaptability and appropriate biases like those transformers provide, such objectives might struggle to produce representations with these properties; without learning objectives specifically designed to harness these architectural properties, the potential for abstract reasoning might remain unrealized. This reciprocal relationship—in which the architectural substrate enables certain training strategies, and these objectives actualize the architectural potential—exemplifies the interplay that lies at the heart of implicit symbolic processing.

Concluding Remarks

The examples throughout this section—from basic architectural biases in CNNs and autoencoders to sophisticated vector programs in transformers—illustrate a spectrum of architectural approaches for inducing symbolic-like characteristics within connectionist frameworks. While not all architectural biases represent equally promising pathways toward advanced reasoning capabilities, they collectively demonstrate that symbolic and continuous processing need not be viewed as inherently incompatible.

However, an important question persists: can vector representations optimally capture the structural

patterns of symbolic reasoning, or might alternative formalisms prove more suitable for certain reasoning tasks? While contemporary deep learning's notable success across diverse domains suggests that high-dimensional continuous spaces offer powerful representational substrates, their adequacy for systematic, compositional reasoning remains an open question. This is particularly true for abstract visual reasoning tasks, where discrete logical operations or hierarchical structures might conceivably be more naturally expressed through alternative representational formalisms. Nevertheless, we believe that vector-based approaches represent a compelling direction worthy of investigation, as they remain fully compatible with gradient-based optimization and modern deep learning infrastructure.

As a foundational investigation of implicit symbolic processing mechanisms, we examine the transformer architecture as the architectural substrate, analyzing how its inductive biases interact with self-supervised property prediction, contrastive learning, and task decomposition. This framework directly instantiates the three hypotheses guiding this research (Sec. 1.2):

- **H1**—that staged decomposition facilitates reasoning—tests whether separating complex reasoning into specialized sub-problems with distinct learning objectives improves systematic reasoning over monolithic approaches. This hypothesis examines the structural dimension of implicit symbolic processing: whether task decomposition itself encourages symbolic-like representations.

- **H2**—that property prediction mitigates dataset biases—tests whether learning objectives that require explicit reasoning about attributes create representations more resistant to spurious correlations than objectives that permit direct answer prediction. This hypothesis examines the synergistic dimension: whether structure-encouraging learning objectives exploiting architectural biases can prevent shortcut learning.

- **H3**—that self-supervised learning suffices for developing systematic reasoning—tests whether the synergy between architectural inductive biases and self-supervised learning objectives can produce reliable reasoning capabilities. This hypothesis examines the training signal dimension: whether implicit symbolic processing capabilities can develop by deriving supervision directly from task structure.

Together, these hypotheses provide empirical validation of the implicit symbolic processing framework. If supported, they offer evidence that combining expressive yet subtly biased architectures with structure-encouraging learning objectives represents a viable pathway toward more resilient, interpretable, and systematically generalizable reasoning in artificial intelligence systems.

Chapter 3

Raven's Progressive Matrices as an Abstract Reasoning Task

3.1 Raven's Progressive Matrices

As introduced in Chapter 1 and discussed in Chapter 2, deep learning models are trained to minimise loss on available data, with no guarantee that the solution they converge on reflects the reasoning process the task was designed to probe (Geirhos et al., 2020). This risk is compounded in abstract reasoning tasks: the output is highly structured relative to a high-dimensional visual input, training sets are limited in scale, and heavily overparameterized models have ample capacity to fit incidental patterns rather than the intended rules (Hastie et al., 2009).

The literature documents numerous failure modes of this kind, where unintended biases undermine reasoning capabilities (Beery et al., 2018; Niven and Kao, 2019). Representations are especially vulnerable to misalignment when training data are limited or contain systematic artifacts. Abstract reasoning tasks represent a particularly challenging regime in this regard, as the available benchmarks have historically been small in scale (Raven, 1936; Zhang et al., 2019a) and susceptible to exploitable statistical regularities (Spratley et al., 2020).

Among the core attributes of general intelligence, abstract reasoning stands out for the breadth of cognitive demands it places on a system: recognizing underlying principles, drawing analogies, and applying relational thinking across novel configurations (Chollet, 2019). At a higher level, this capacity extends to conceptualization—forming, manipulating, and deploying abstract representations to generate plans and original solutions.

Sequential pattern completion is one activity that puts these demands into practice and has become especially prominent as a benchmark in the research community. It requires a solver to infer the rules governing an observed sequence and extrapolate them to unseen data. John C. Raven formalized this principle into a visual intelligence test in the 1930s (Raven, 1936), an instrument now collectively referred

to as Raven's Progressive Matrices (RPMs).

An RPM problem (Figure 1.1) consists of a 3×3 grid containing eight context panels with *arrangements* of 2D geometric *objects*. The objects conform to rules that govern inter-panel relationships—for instance, a progression in the number of objects or a logical operation on their presence (Zhang et al., 2019a). Solving the matrix requires selecting, from eight answer candidates, the panel that completes those rules, which demands simultaneously disentangling regularities and recognizing the analogies that hold across them (Carpenter et al., 1990).

3.1.1 The RAVEN Dataset

The original collection of RPM problems, known as Standard Progressive Matrices (Raven, 1936), consisted of only 60 tasks—insufficient for effectively training data-hungry machine learning models. Consequently, researchers have developed larger datasets and task generators, with **RAVEN** (Zhang et al., 2019a) [1] (Figure 3.1) and **I-RAVEN** (Hu et al., 2021) [2] being among the most popular currently. These resources have become essential for training and evaluating modern machine learning models due to their increased complexity and capacity to challenge AI systems.

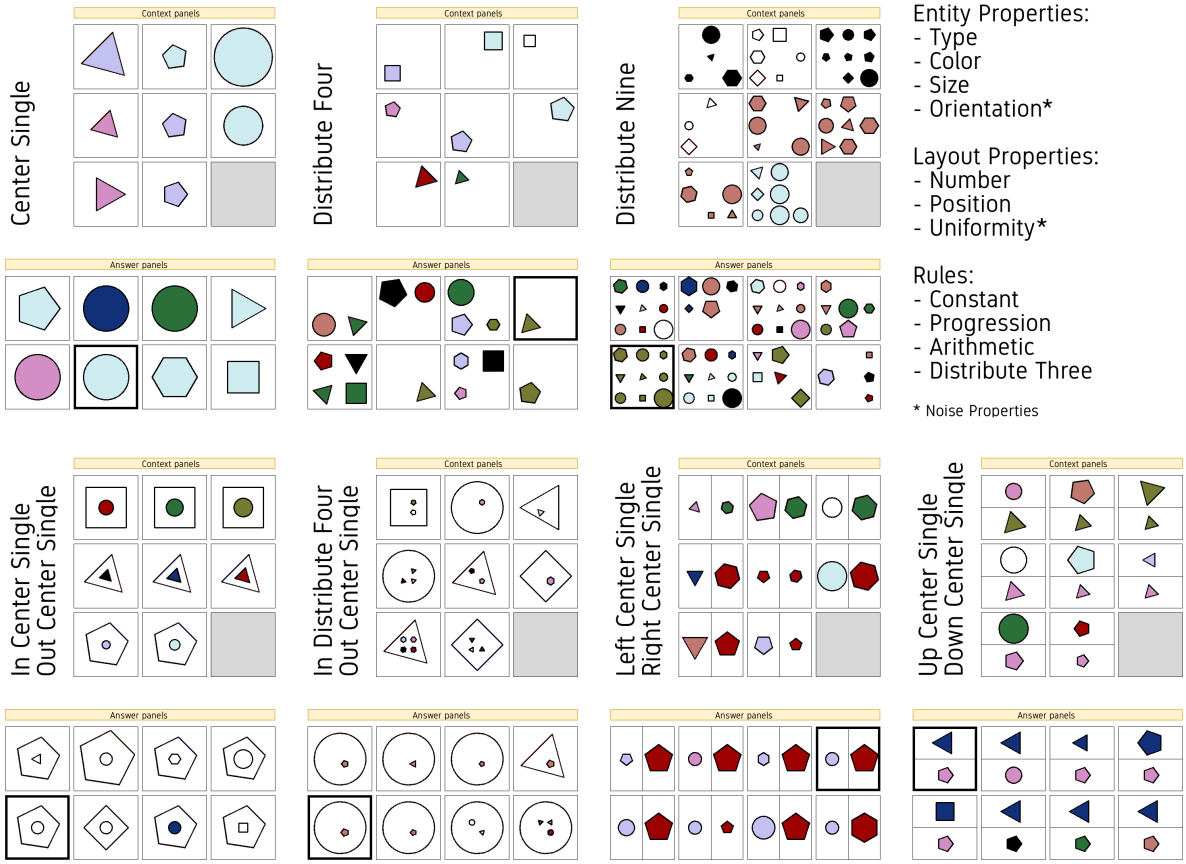


FIGURE 3.1: Possible arrangement patterns in the RAVEN dataset, showing spatial configurations of objects within panels. Throughout this work, context panels are referred to as c1–c8 (numbered left-to-right, top-to-bottom) and answer panels as p1–p8.

The RAVEN dataset follows the common approach of RPM benchmarks, where tasks are generated from symbolic specifications expressed in an *image description grammar* that captures visual concepts such as position, shape type, color, number of objects, and inside-outside relationships. The image

[1] `https://wellyzhang.github.io/project/raven.html#dataset`
[2] `https://github.com/husheng12345/SRAN`

description grammar hierarchically defines visual concepts and their relationships with the following levels: Scene, Structure, Component, Layout, and Entities, as summarized in (Zhang et al., 2019a) (p. 3):

> "Specifically, the A-SIG for RPM has 5 levels—Scene, Structure, Component, Layout, and Entity. Note that each grammar level could have multiple instantiations, i.e., different categories or types. The Scene level could choose any available Structure, which consists of possibly multiple Components. Each Component branches into Layouts that links Entities. Attributes are appended to certain levels; for instance, (i) Number and Position are associated with Layout, and (ii) Type, Size, and Color are associated with Entity. Each attribute could take a value from a finite set. During sampling, both image structure and attribute values are sampled. To increase the challenges and difficulties in the RAVEN dataset, we further append 2 types of noise attributes—*Uniformity* and Orientation—to Layout and Entity, respectively. Uniformity, set false, will not constrain Entities in a Layout to look the same, while Orientation allows an Entity to self-rotate."

The grammar defines seven spatial arrangements, each containing at least one object, with the maximum number of objects as follows:

- center-single (1),
- distribute-four (4),
- distribute-nine (9),
- in-center-single-out-center-single (2),
- in-distribute-four-out-center-single (5),
- left-center-single-right-center-single (2),
- up-center-single-down-center-single (2).

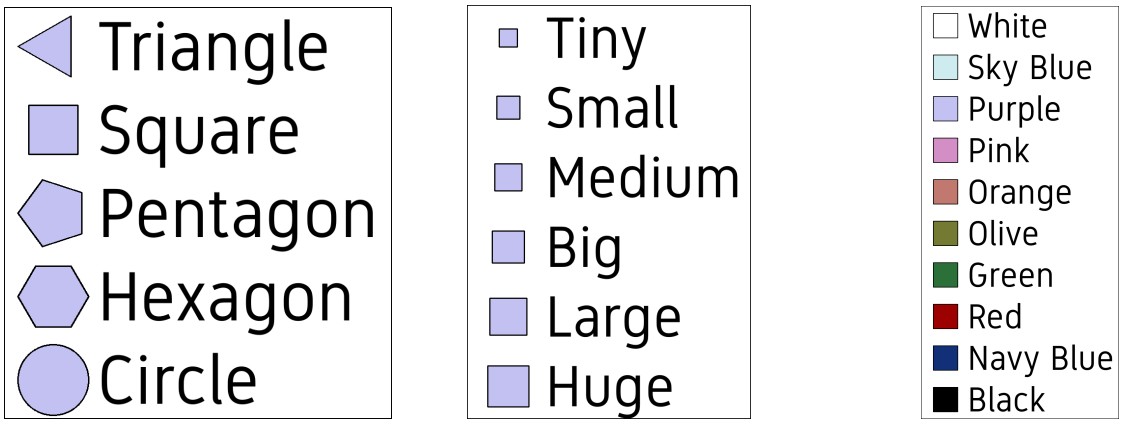
Figure 3.1 presents, for each of the seven spatial arrangements, an example task from the RAVEN dataset.

3.1.2 Visual Properties and Pseudocoloring

Figures (entities) in RAVEN are described by four properties: Type, Size, Color, and Orientation. Each property has a specific set of possible values:

- **Type**: triangle, square, pentagon, hexagon, circle (5 values; see Figure 3.2a),
- **Size**: 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 (6 values, assigned descriptive names *tiny*, *small*, *medium*, *big*, *large*, and *huge*; see Figure 3.2b),
- **Color**: 255, 224, 196, 168, 140, 112, 84, 56, 28, 0 (10 grayscale values, enhanced with pseudocoloring in this thesis; see Figure 3.2c),
- **Orientation**: -135°, -90°, -45°, 0°, 45°, 90°, 135°, 180° (8 values).

The Color property follows a specific ordering, rendered in grayscale by default. However, for analyzing task properties, rules, and patterns, grayscale representation is suboptimal as it makes distinguishing between different values challenging. We employ **pseudocoloring** utilizing a colormap from the Matplotlib library. Since the standard Matplotlib colormap may still make some color pairs difficult to distinguish



(A) Object type (B) Size (C) Color

FIGURE 3.2: RAVEN attribute examples ordered by increasing number of classes: object type, size, and color.

when applied to RAVEN data, we enhance distinguishability by merging three colormaps: taking the first seven colors from "cubehelix," the next from "hot," and the last two from "gist_earth." We assign common color names to the pseudocolored values (e.g., *red*, *orange*, *yellow*), significantly improving visual distinguishability. Similarly, we assign descriptive names to Size values (*tiny* through *huge*), facilitating more effective description and analysis of task patterns throughout this work.

The Orientation property is treated as noise and considered irrelevant in the original **RAVEN** dataset design, a deliberate choice by the authors (Zhang et al., 2019a). Consequently, this thesis follows the same convention and does not include Orientation in the property representation.

3.1.3 Rules and Pattern Structure

In the **RAVEN** dataset, patterns are created by rules applied to panel properties. Rules are applied only row-wise (Zhang et al., 2019a), unlike in other RPM datasets such as **Procedurally Generated Matrices (PGM)** (Barrett et al., 2018). Each property has its own set of rules, which can be as simple as constancy (the property remains unchanged across panels in a row) or more complex patterns.

There are four types of rules:

- Constant: Properties do not change, except in special cases.
- Progression: Property values increase or decrease by a constant amount.
- Arithmetic: Property values are governed by arithmetic operations (addition or subtraction).
- Distribute-Three: Property values are distributed among all cells in a row; the set of values remains constant across rows.

In addition to appearance properties of figures (Type, Size, Color, and Orientation), the content of context panels is also determined by layout properties (Number, Position, and Uniformity). However, not all properties have rules applied to them. Orientation is treated as noise and considered irrelevant in RAVEN; thus, rules are not applied to it. Uniformity is a flag that affects how properties manifest rather than an attribute to which rules are directly applied. Consequently, rules operate on five attributes: the three appearance properties (Type, Size, and Color) and the two layout properties (Number and Position).

Zhang et al. (2019a) note that Number and Position are strongly coupled, and that the Arithmetic rule on Type is excluded as counterintuitive, resulting in 19 rule-attribute combinations rather than 20 (supplementary material, p. 2):

> "Among all attributes, we realize that Number and Position are strongly coupled, hence we do not allow non-Constant rules to co-occur on both attributes. With 4 rules and 5 attributes, we could have had 20 rule-attribute combinations. However, we exclude Arithmetic on Type, as it is counterintuitive, resulting in 19 combinations in total."

The coupling between Number and Position has a subtle practical implication: even when the Number attribute follows a Constant rule, a non-Constant rule on Position can cause the effective number of objects to change, and vice versa. Only when both are Constant will neither attribute vary. A further source of apparent change in a Constant attribute is noise introduced when the Uniformity flag is set to False. Zhang et al. (2019a) define the Constant rule as follows (supplementary material, p. 2):

> "Constant: Attributes governed by this rule would not change in the row. If it is applied on Number or Position, attribute values would not change across the three panels. If it is applied on Entity level attributes, then we leave 'as is' the attribute in each object across the three panels. This design would render every object the same if Uniformity is set to True; otherwise, it will introduce noise in a problem instance."

For detailed descriptions of rule behavior for each property, see the supplementary material of Zhang et al. (2019a).

3.1.4 Uniformity and Object Persistence

The RAVEN documentation defines a binary *uniformity* flag that controls whether property values can vary within panels (Zhang et al., 2019a). However, through systematic analysis of the dataset's generation mechanism (the source code of the authors' implementation) and rule interaction patterns, we identify that uniformity in RAVEN actually manifests in three distinct forms: *panel uniformity*, *object persistence*, and *row uniformity*. These types, while not explicitly defined by the dataset creators, characterize different aspects of property consistency across the task structure and prove essential for rule inference.

Panel Uniformity refers to whether all objects within a single panel share identical appearance properties. This occurs when either: (1) the uniformity flag is set to `True`, or (2) non-constant rules are applied to appearance properties, which enforces uniformity regardless of the flag setting.

Object Persistence refers to whether objects maintain their identity across panels in a row, enabling one-to-one correspondence despite transformations. Consider a panel containing three objects with different colors: red, blue, and green. Such a panel lacks uniformity. However, when examining subsequent panels in the same row, there is no immediate certainty that all colors will remain the same, or even that the same number of objects will persist. Depending on the applied rules, there may exist a form of object equivalence—a one-to-one relationship that we refer to as object persistence. Objects can change their positions and certain appearance properties according to non-constant rules; however, these changes are not random, as orientation remains the same and the constant properties remain unchanged. Object persistence occurs when a constant rule is applied to the Number property and an Arithmetic rule is not applied to the Position property. Under these conditions, objects can be tracked across panels despite undergoing transformations, maintaining their identity through the properties that remain constant.

Row Uniformity indicates whether the set of appearance property values remains consistent across all panels in a row. When object persistence does not exist and panel non-uniformity conditions are met, appearance property values may be resampled for each panel in the row, resulting in row non-uniformity. In such cases, entirely different sets of property values appear in successive panels, breaking the visual continuity of the row. ■

TABLE 3.1: Conditions determining panel uniformity and object persistence in RAVEN. Row uniformity exists whenever at least one of these holds.

Property	**Conditions**
Panel Uniformity	Uniformity flag set OR: Non-constant rules on appearance properties
Object Persistence	Number rule is Constant AND: Position rule is NOT Arithmetic
Derived property:	
Row Uniformity	Panel Uniformity OR Object Persistence

Figure 3.3 illustrates these three types of uniformity. In example (a), although the uniformity label is not set, non-constant rules applied to all appearance properties enforce uniformity within each panel. Example (b) demonstrates object persistence: despite objects occupying different positions and exhibiting different types and sizes due to applied rules, a one-to-one relationship is maintained. Objects can be tracked by their color and orientation, even though matching by orientation alone is challenging since different object types depict orientation differently. Abstractly, these represent the same objects undergoing systematic transformations rather than independent entities. Example (c) shows a scenario without object persistence due to the applied Arithmetic rule on position. In this case, subsequent panels in the row contain different objects, resulting in row non-uniformity where property values are resampled for each panel.

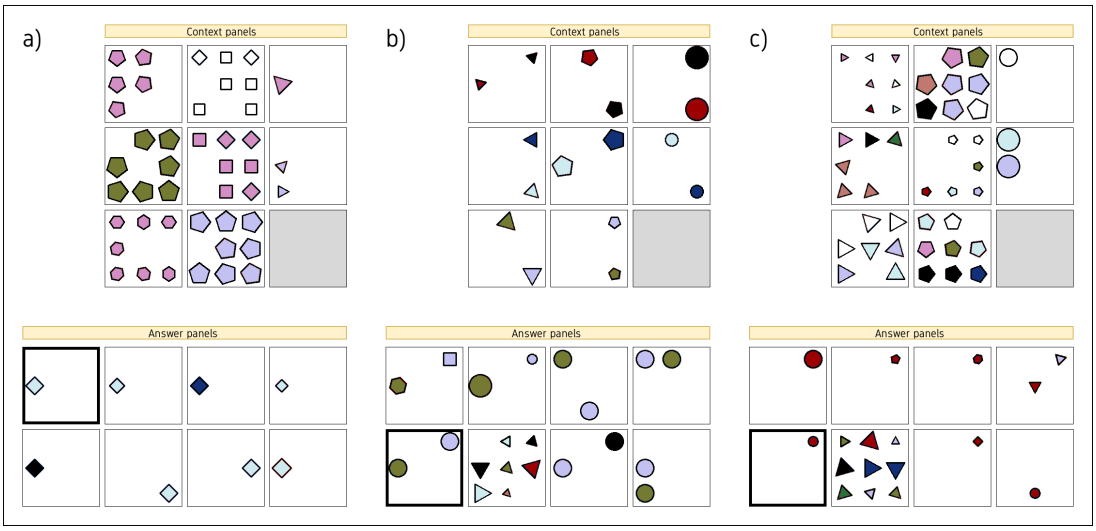


FIGURE 3.3: Three distinct types of uniformity we identify in the RAVEN dataset through analysis of rule interactions: (a) panel uniformity despite non-constant appearance rules, (b) object persistence with panel non-uniformity, and (c) row non-uniformity without object persistence.

The conditions producing each uniformity type emerge from interactions between the rule choices and the uniformity flag. Panel uniformity arises when the uniformity flag is set *or* when non-constant rules apply to appearance properties. Object persistence requires a constant Number rule combined with non-Arithmetic Position rules. Critically, row uniformity exists whenever *either* panel uniformity *or* object persistence holds—it is absent only in the specific case where both are lacking (constant appearance rules with uniformity flag off *and* either varying Number or Arithmetic Position rules). Table 3.1 formalizes these patterns.

Understanding these uniformity types is important for analyzing how rules operate in RAVEN tasks, particularly during rule inference. Consider the challenge of determining which rule governs the Position

attribute: when objects move between panels within a row, one must first establish object correspondence—which object in panel i corresponds to which object in panel $i+1$. This matching relies on tracking objects through their *constant* appearance attributes (Type, Size, Color). When all three appearance attributes undergo transformations governed by non-constant rules, only the Orientation attribute remains available for establishing object identity across successive panels. However, this tracking is only possible when *object persistence* exists. Without object persistence, objects in successive panels are entirely independent entities. Thus, detecting whether object persistence holds is not merely a taxonomic exercise—it determines which analytical strategies apply to rule inference.

3.1.5 Meta-Target Encoding

The RAVEN dataset provides *supplementary annotation systems* beyond raw images and the index of the correct answer: each task includes metadata specifying which rules apply to which attributes, along with structural information about the task's hierarchical organization (scene type, arrangement pattern, etc.). These annotations enable multi-task learning approaches where models simultaneously predict rules alongside solving the RPM task itself.

RAVEN encodes this metadata using two complementary representations: `meta_matrix` and `meta_target`. The `meta_matrix` preserves complete rule-attribute pairings through row-wise one-hot encoding, where each row specifies which rule applies to which attribute(s). For example, a row encoding [0, 1, 0, 0, 0, 0, 1, 0, 0] indicates that a Progression rule applies to the Type attribute. The `meta_target` aggregates all rows via element-wise OR operations, producing a compact representation that indicates which rules and attributes appear in the task but loses the specific pairings. This compression trades detailed relational information for a fixed-dimensional vector suitable for multi-label classification.

This encoding mechanism adapts the meta-target framework introduced in "Measuring abstract reasoning in neural networks" (Barrett et al., 2018), which represents rules as tuples (r, o, a) encompassing rule types, object classifications, and target attributes. RAVEN simplifies this to (r, a) tuples—omitting object classifications since all entities are geometric figures rather than diverse object categories. Additionally, RAVEN provides hierarchical task descriptions through structured `meta_structure` vectors and human-readable XML representations, enabling systematic task analysis beyond rule prediction alone.

Listing 3.1 illustrates the complete metadata for task 1 in the RAVEN training split (visualized as example (a) in Figure 3.1). The `meta_matrix` reveals that four rules govern this task: Constant applies to Number and Position jointly, Progression applies to Type, and Arithmetic applies independently to both Size and Color. The `meta_target` aggregates these into a single vector indicating the presence of rules and attributes, while the `meta_structure` and `structure` fields encode the hierarchical task organization (Grid arrangement with Center-Single pattern).

```
target    <- index of the correct answer panel (0-7)
5

predict   <- answer index predicted by the heuristic solver
5

image
[[[255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  [255 255 255 ... 255 255 255]
  ...
```

```
meta_matrix
[[1 0 0 0 1 1 0 0 0]  <- Row 1: Constant rule + Number & Position attributes
 [0 1 0 0 0 0 1 0 0]  <- Row 2: Progression rule + Type attribute
 [0 0 1 0 0 0 0 1 0]  <- Row 3: Arithmetic rule + Size attribute
 [0 0 1 0 0 0 0 0 1]  <- Row 4: Arithmetic rule + Color attribute
 [0 0 0 0 0 0 0 0 0]  <- Unused row
 [0 0 0 0 0 0 0 0 0]  <- Unused row
 [0 0 0 0 0 0 0 0 0]  <- Unused row
 [0 0 0 0 0 0 0 0 0]] <- Unused row

meta_structure
[1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0]

meta_target
[1 1 1 0 1 1 1 1 1]

structure
[b'Scene' b'Singleton' b'Grid' b'Center_Single' b'/' b'/' b'/' b'/']
```

Listing 3.1: Example metadata from RAVEN task #1 (training set). The `target` field stores the index of the correct answer panel (0–7). The `predict` field stores the answer predicted by the heuristic constraint-satisfaction solver described in the original paper (Zhang et al., 2019a), which achieves perfect accuracy by checking which candidate satisfies the most constraints given the symbolic representation of the task. The `meta_matrix` encodes four rule-attribute pairs. Element-wise OR aggregation produces the `meta_target`, which indicates that Constant, Progression, and Arithmetic rules appear, affecting Number, Position, Type, Size, and Color—but does not specify which rule applies to which attribute.

3.2 Bias in RAVEN Datasets

3.2.1 Similarity Bias

Constructing answer sets for RPM tasks presents fundamental challenges for dataset designers (Zhang et al., 2019a; Spratley et al., 2020). A naive approach to constructing RPM answer sets involving random panel sampling introduces a problematic pattern: since context panels naturally share visual features due to rule-based transformations, the correct answer panel usually resembles context panels more closely than randomly generated distractors would. This creates an exploitable pattern where models can achieve high performance by selecting the answer most similar to context panels without engaging in genuine abstract reasoning about the governing rules. We term this *similarity bias*—a form of *shortcut learning* where models exploit surface-level resemblances between inputs and outputs rather than learning the underlying abstract relationships. The severity of this bias depends significantly on how answer sets are constructed—careful design can mitigate but not entirely eliminate this challenge without compromising task difficulty.

This phenomenon represents a broader challenge in machine learning where implicit dependency between input and output enables models to achieve favorable performance metrics without learning the intended task. Rather than learning the underlying task structure, models may learn to replicate or match input patterns through superficial feature comparison. Such issues manifest across diverse domains. In *time series forecasting models* (Makridakis et al., 2020; Hyndman and Athanasopoulos, 2018), future

values are predicted by simply outputting the last known point $f(n) = f(n-1)$. Since time series data frequently exhibit temporal autocorrelation where neighboring observations are similar, this approach may lead to an overestimation of the predictor's performance while failing to accurately forecast future trends. Similar situations arise in *image-to-image models* (Isola et al., 2017; Zhu et al., 2017) designed to modify images according to specified objectives, such as style transfer or image segmentation (Gatys et al., 2016; Long et al., 2015). This phenomenon is particularly pronounced in tasks requiring minimal input adjustments, including image restoration, artifact correction, and image inpainting (Pathak et al., 2016; Liu et al., 2018), which essentially perform reconstruction with minor modifications. These models tend to favor the simplest solutions by producing outputs nearly identical to the inputs, thereby circumventing the challenge of learning the actual transformation process.

Quantifying Structural Property Overlap

We demonstrate the presence of similarity bias in the RAVEN dataset by examining the correspondence between property values of answer panels and context panels. We quantify similarity as the fraction of property values in an answer panel that appear in at least one context panel: for each property (Type, Size, Color), we check whether the answer panel's value matches any value present among the eight context panels, then average this fraction across all properties. Table 3.2 presents this analysis for the *center-single* arrangement.

Table 3.2: Property overlap between answer and context panels (RAVEN, center-single).

Answer Type	**Overlap (%)**
Correct Answer	68.10
Incorrect Answers	37.14

Table 3.3: Distance between c8 and answer panels (RAVEN).

	Hamm. (%)	**Absolute**
Correct Answer	68.70	1.56
Incorrect Answers	75.07	1.82
Δ (margin)	6.37	0.26

For the RAVEN test set, the average overlap reaches 68.10% for correct answer panels, compared to only 37.14% for incorrect answer panels, confirming the presence of structural similarity. While these statistics alone do not prove the existence of a *strategy* capable of solving benchmarks based solely on similarity computation, there are a number of ways in which these characteristics can be exploited to provide a superficial shortcut that enables high performance without reasoning about the underlying rules.

We label context panels c1–c8 and answer panels p1–p8 as defined in Figure 3.1. A hypothetical similarity-based strategy could involve selecting the answer panel that most closely resembles one of the context panels. Within the context, the *query panel* typically shares similarities with other panels, particularly with panel c8 positioned directly to its left, since the answer for the query panel should represent a logical continuation of c8. For instance, applying only one non-constant rule results in the query panel differing by merely one property from c8, with the differing property likely exhibiting a similar value. Consequently, c8 provides information about the query panel, suggesting that selecting the panel most similar to it or to all context panels could constitute a viable strategy. To assess whether such a strategy is viable, we conducted detailed distance analysis between context panels and answer candidates.

Our structural analysis, detailed in Appendix A.3.3, demonstrates that the discrepancy between the query panel and panel c8 is indeed the smallest among all context panels. As shown in Table 3.3, the correct answer differs from c8 by an average of only 1.56 property values (68.70% Hamming distance) in RAVEN, while other context panels exhibit differences around 2.0 (see complete distance matrix in Appendix Table A.15). More importantly, incorrect answer panels differ from c8 by merely 1.82 property

values (75.07% Hamming distance)—a discrimination margin of only 0.26 property values. This narrow gap creates considerable ambiguity for similarity-based strategies, demonstrating that while c8 provides strong cues about the correct answer, distinguishing it from carefully constructed incorrect answers requires minimal but precise differences.

Beyond selecting the answer most similar to a single context panel, performance of similarity-based heuristics could potentially be enhanced by devising strategies that rely on simple cues rather than abstract reasoning. For instance, if context panels c3 (top right) and c6 (middle right) appear similar, selecting the answer most similar to c6 could suggest similarity across the entire column. Similarly, if c7 (bottom left) and c8 exhibit resemblance, choosing the answer most similar to c8 may indicate similarity among objects in the row.

Figure 3.4 illustrates the effectiveness of this heuristic across tasks of varying complexity. In example **(a)**, three rules are applied: *progression* on type, *progression* on color, and an *arithmetic* rule on size. However, rather than inferring these rules from the entire context, the solver may directly select answer panel p2 (the huge olive circle), since it is identical to c6. Panel c3 is also a huge circle, suggesting that this column should contain only huge circles.

In example **(b)**, each row of context panels features the same object with only one non-constant rule applied: *distribute three* on color. The solver can bypass inferring this rule and simply choose p7 (the tiny purple hexagon), since the entire bottom row contains such hexagons, and among the two tiny hexagons in the answer set, only the purple one has a color appearing in the context.

Similarity bias can also be exploited in more complex tasks, such as example **(c)**, governed by two rules: *progression* of type (consecutive panels in each row contain figures with increasing numbers of edges, with circles corresponding to infinite edges) and *progression* of object count. Nevertheless, the straightforward approach is to select the answer panel containing circles, since panels in the last column (c3 and c6) both feature circles, and only one answer panel contains circles. Beyond being mathematically most similar to context images, these panels represent intuitive solutions for humans—answers that appear correct at first glance without deliberation.

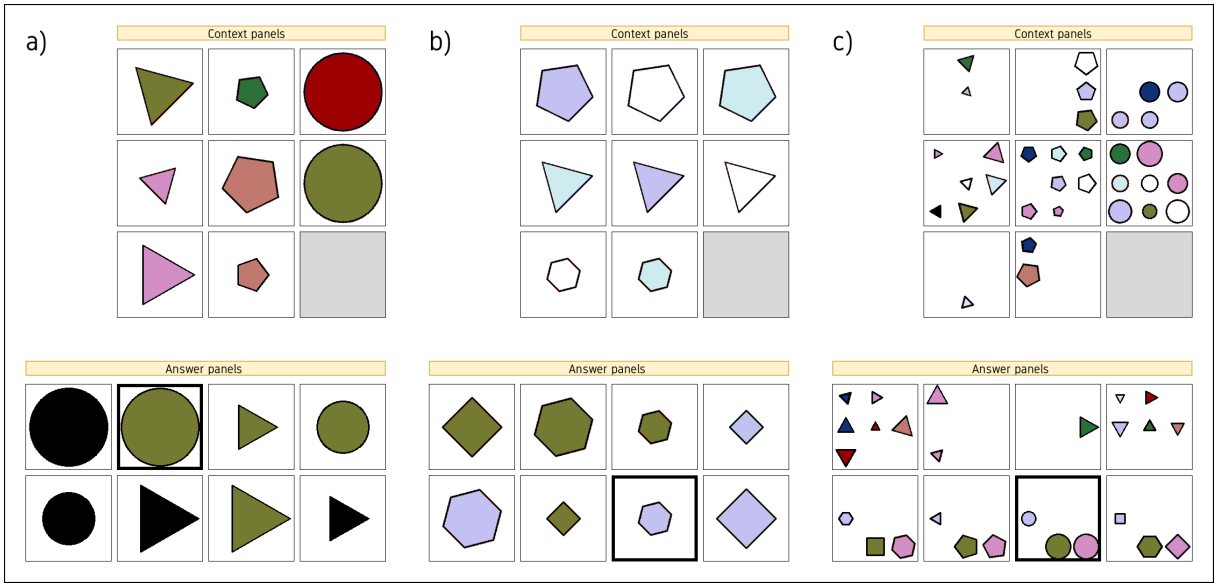


FIGURE 3.4: Examples of tasks featuring *similarity bias* where solvers can exploit visual similarities between context and answer panels rather than inferring abstract rules (RAVEN dataset).

Extensions to this heuristic could search for an answer where the difference from c6 approximately equals the difference between c3 and c6. This approach could work well for constant and progression rules, potentially serving as an approximation for arithmetic and distribute three rules. Deep neural

networks can expand this heuristic by integrating more sophisticated patterns difficult for humans to perceive and learning the contexts where these patterns prove most effective.

RAVEN's Mitigation Strategy

To prevent reliance on the similarity-based shortcuts discussed above, the creators of the RAVEN dataset proposed a specific answer set construction strategy: incorrect answers are designed to closely resemble the correct answer, typically differing by only one property (Zhang et al., 2019a). This approach complicates similarity-based answer selection. Given this design and the c8-query similarity documented above (discrimination margin of only 0.26 property values), distinguishing correct from incorrect answers through similarity alone becomes substantially more challenging.

To assess whether this mitigation strategy successfully prevents similarity-based shortcuts, we developed simple solvers that select answers based purely on similarity, with no reasoning about rules or patterns. These solvers operate across three *representation levels*:

- **Pixel-level similarity**: Direct comparison of raw images using standard metrics (Mean Squared Error, Relative Average Spectral Error, etc.),
- **ImageNet-pretrained features (frozen)**: Similarity computed in the feature space of an ImageNet-pretrained **EfficientNetV2** (Tan and Le, 2021) backbone, representing baseline perceptual similarity from general-purpose visual features,
- **RAVEN property-adapted features**: Similarity in the feature space of EfficientNetV2 after adaptation through RAVEN property prediction training (see Section 4.3), revealing whether task-specific learning amplifies exploitable patterns.

For each representation level, we evaluate multiple context aggregation strategies—the primary experimental setting determining which context panels the solver uses for comparison:

- **c6**: Compare answer candidates only to panel c6 (middle-right context panel),
- **c8**: Compare answer candidates only to panel c8 (bottom-middle context panel, immediately left of the query),
- **c6+c8**: Combined comparison to panels c6 and c8,
- **Mean context**: Average similarity to all eight context panels,
- **Heuristic**: A rule-based strategy that first determines which context panel to use as the comparison reference by measuring within-context similarity: if c3 and c6 (the right column) are more similar to each other than c7 and c8 (the bottom row), the heuristic selects c6 as the reference panel; otherwise it selects c8. The answer candidate most similar to the chosen reference is then selected. Complete definition provided in Appendix A.3.1.

Each strategy is evaluated using various similarity metrics as measurement tools. For each combination of representation level and context strategy, the solver selects the answer panel that minimizes the distance (or maximizes similarity) to the chosen context reference. Table 3.4 reports the accuracy (the percentages of correct predictions) for the representative strategy–metric combinations. Complete results for all strategies and all metrics are provided in Appendix A.3.1–A.3.2 (Tables A.5–A.13).

All strategies perform below the random baseline consisting of sampling uniformly from the answer panels (12.5%), with the best pixel-based heuristic achieving only 9.34% accuracy. Notably, RAVEN

TABLE 3.4: Accuracy (%) achieved by similarity-based heuristic solvers on RAVEN across three representation levels. Each row shows a similarity metric (measurement tool) with its associated context aggregation strategy in parentheses. The strategy determines which context panels are compared to answer candidates; the metric specifies how similarity is measured. "Best X" rows report the highest-performing strategy–metric combination within each representation level. All strategies perform below random baseline (12.5%), validating RAVEN's answer set construction. Complete per-strategy breakdowns provided in Appendix A.3.1–A.3.2.

Metric & Strategy	**Accuracy (%)**
Pixel-level similarity:	
MSE (Mean context)	6.44
RASE (c6+c8)	9.34
Best pixel-level	9.34
ImageNet-pretrained features (frozen):	
Cosine similarity (Mean context)	5.68
MAE (c6+c8)	8.46
Best frozen features	8.81
RAVEN property-adapted features:	
Cosine similarity (Mean context)	5.17
MAE (c6+c8)	7.59
Best adapted features	7.64
Random Baseline	**12.5**

property-adapted features perform *worse* than frozen ImageNet features (7.64% vs 8.81%), suggesting that property prediction training actively suppresses similarity shortcuts on RAVEN rather than amplifying them. These results validate the effectiveness of RAVEN's answer set construction method in mitigating simple similarity exploitation.

Comprehensive experimental details, including additional metrics (UQI, ERGAS, SAM, PSNR-B) and complete per-strategy breakdowns across all context aggregation approaches, are provided in Appendix A.3.1 (pixel-level analysis), Appendix A.3.2 (frozen and adapted feature analysis), and Appendix A.3.3 (complete spatial distance matrices for all context panel positions).

3.2.2 Answer Set Bias: The Paradox of Bias Mitigation

Paradoxically, the strategy employed to mitigate similarity bias—making incorrect answers differ by only one property from the correct answer—unintentionally introduces a different form of bias. This method of constructing alternative answers creates an opportunity for building a straightforward shortcut: selecting the answer whose characteristics are overall most similar to other answers: more informally, choosing the answer panel that appears "most average" among all answer panels. Quite often, identifying the correct answer panel requires finding the modal value for each property among the values present in the answer set, i.e., the value that is most common in the answer set.

The examples in Figure 3.5, taken from the RAVEN test set, illustrate this *answer set bias*, where correct answer panels can be identified by examining property distributions across answer panels and selecting the answer featuring the most common properties (Spratley et al., 2020). In example **(a)**, each panel in the answer set features one figure in the upper-right corner, with the following property distributions:

- *type*: 7 squares, 1 triangle,
- *color*: 4 red figures, 1 green, 1 black, 1 olive, 1 sky blue,
- *size*: 7 big figures, 1 small.

By selecting the most frequently occurring properties (i.e., the modal value of each marginal distribution), we choose the panel with a big red square (marked with a darker frame), which is the correct answer for this task.

Similar reasoning applies to more demanding tasks with multiple figures, as demonstrated in example **(b)**. When analyzing this visualization, we observe that olive is the most frequently used color in the answer set, the most common figure pair is a pentagon and a square, and in 7 out of 8 answer panels, both figures are positioned along the right edge. This again allows selecting the correct answer despite the task's complexity, which involves four-figure arrangements, three rules, and positional operations.

The same strategy enables solving example **(c)**, which features a layout with two independent insets placed vertically. In the lower inset, all figures in the answer set are large navy blue figures, with nearly all being squares. In the upper inset, most figures are green triangles, with 7 sharing the same size. The majority rule identifies the correct answer without analyzing the context panels.

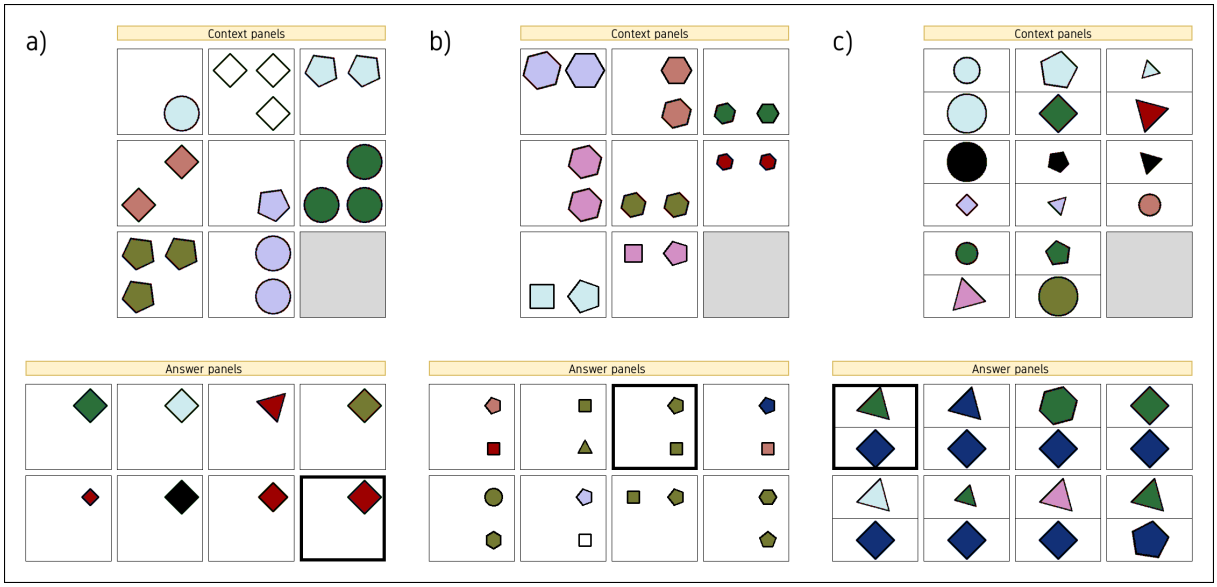


FIGURE 3.5: Examples of tasks featuring *answer set bias* where correct answers can be identified by analyzing property distributions across answer panels without considering context (RAVEN dataset).

This limitation of RAVEN was particularly highlighted by Spratley et al. (2020); Hu et al. (2021); Benny et al. (2021), who demonstrated that *context-blind models*—operating without access to context panels—can exceed 90% accuracy on RAVEN, confirming the severity of this bias.

Uniformity Bias: A Specific Manifestation

Figure 3.6 shows examples of yet another, particularly striking form of bias we identified in RAVEN and termed *uniformity bias*. In these tasks, the most visually salient feature of context panels is that each of them contains multiple copies of the same object (identical size, color, and type; recall that rotation angle is irrelevant in RAVEN). Since only one answer panel exhibits this kind of uniformity, it becomes the obvious first choice, without requiring discovery of the rules that actually govern the context panels.

For instance, in example **(a)**, the actual reasoning expected from an 'honest' solver involves recognizing that (i) the number of objects in the right-hand panel of each row equals the total number of objects in the left and central panels, (ii) the shape should be triangular because it is absent from the last row, and (iii) the objects should be tiny. This explanation, however correct, can unfortunately be completely bypassed by shortcut reasoning, given the more obvious uniformity of figures in the context panels and its absence in all but one of the answer panels.

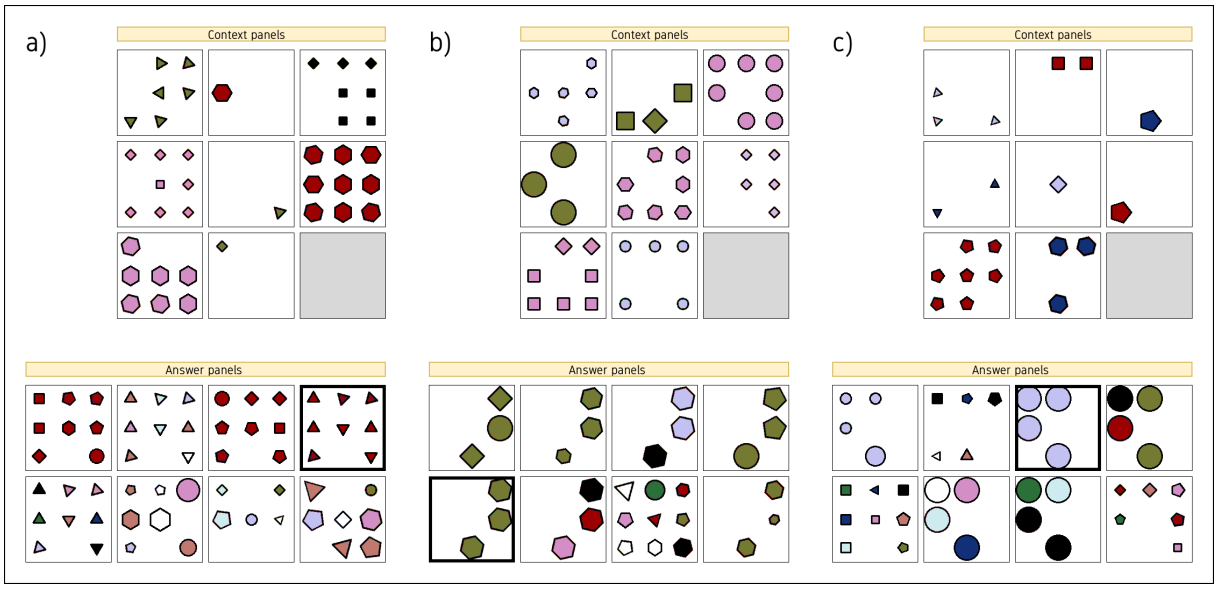


FIGURE 3.6: Examples of tasks featuring *uniformity bias*, a specific form of answer set bias where correct answers can be identified by the unique presence of uniform objects matching context panel patterns (RAVEN dataset).

This bias arises directly from the uniformity flag being set to false in answer panels. As described in Section 3.1.4, context panels can still exhibit panel uniformity even with this flag set to false when non-constant rules are applied to appearance properties (Type, Size, Color)—the rule application enforces uniformity regardless of the flag. This creates a visual mismatch: context panels display uniform objects while most answer panels contain non-uniform objects, making the single uniform answer panel immediately identifiable as the correct choice.

3.2.3 Improved Benchmarks: I-RAVEN and RAVEN-FAIR

To address the severe answer set biases documented above—which enable context-blind accuracy above 90% (Spratley et al., 2020; Hu et al., 2021; Benny et al., 2021)—researchers developed improved benchmarks with more sophisticated answer generation procedures. **I-RAVEN** (Hu et al., 2021) employs an Attribute Bisection Tree (ABT) algorithm to generate answer sets, completely eliminating answer set bias. **RAVEN-FAIR** (Benny et al., 2021) (which we abbreviate as **F-RAVEN**) uses a graph-based perturbation process where incorrect answers form a connected graph with edges representing single-attribute changes, substantially reducing context-blind performance to 17.2% (Benny et al., 2021). Both benchmarks thus provide more reliable evaluation of abstract reasoning when answer panels are accessible during training, though with different degrees of bias mitigation.

To assess whether these improvements introduce other biases, we extended the similarity bias experiments from Section 3.2.1 to include I-RAVEN and F-RAVEN. Tables 3.5–3.7 present the complete cross-benchmark analysis, revealing the trade-offs inherent in answer set construction strategies.

TABLE 3.5: Average percentage overlap between answer panel properties and context panel properties for the center-single arrangement across RAVEN benchmarks. High overlap for correct answers (68–70%) indicates structural similarity, while lower overlap for incorrect answers (25–37%) reflects answer set construction strategies to mitigate bias. This table extends Table 3.2 with I-RAVEN and RAVEN-FAIR data.

Benchmark	Correct Answer (%)	Incorrect Answers (%)
RAVEN	68.10	37.14
I-RAVEN	69.65	25.81
F-RAVEN	68.80	25.01

Property overlap patterns remain consistent across benchmarks: approximately 69% for correct answers. However, incorrect answer overlap decreased substantially (25–26% for I-RAVEN and RAVEN-FAIR vs 37% for RAVEN), creating a larger gap between correct and incorrect answer overlap.

Table 3.6 extends the c8 distance analysis to all three benchmarks. Both property overlap gaps and c8 distance metrics reveal the same pattern: I-RAVEN and RAVEN-FAIR show larger discrimination margins between correct and incorrect answers. The c8 distance gap increased from 0.26 property values in RAVEN to 0.42–0.43 in the improved benchmarks, while the property overlap gap widened from 32 to 44 percentage points. These larger margins mean correct answers are more distinguishable from incorrect ones based on similarity to context panels.

Table 3.6: Mean distance between context panel c8 (bottom-middle) and answer panels across RAVEN benchmarks. Hamming Distance shows percentage of differing properties out of 101-dimensional vectors. Absolute Difference shows mean number of property value mismatches. Larger Δ values indicate greater discrimination margin between correct and incorrect answers, making similarity-based strategies more viable. This table extends Table 3.3 with I-RAVEN and RAVEN-FAIR data.

	Hamming Distance (%)			**Absolute Difference**		
Benchmark	**Correct**	**Incorrect**	**Δ**	**Correct**	**Incorrect**	**Δ**
RAVEN	68.70	75.07	6.37	1.56	1.82	0.26
I-RAVEN	69.73	79.80	10.07	1.60	2.02	0.42
F-RAVEN	69.62	80.86	11.24	1.61	2.04	0.43

Table 3.7: Task accuracy (%) achieved by similarity-based heuristic solvers across RAVEN benchmarks. Each row shows a similarity metric (measurement tool) with its associated context aggregation strategy in parentheses. The strategy determines which context panels are compared to answer candidates; the metric specifies how similarity is measured. "Best X" rows report the highest-performing strategy–metric combination within each representation level. RAVEN maintains all strategies below random baseline (12.5%), while I-RAVEN and RAVEN-FAIR show modest vulnerability to learned similarity exploitation. This table extends Table 3.4 with I-RAVEN and RAVEN-FAIR data; complete per-strategy results in Appendix A.3.1–A.3.2.

Metric & Strategy	**RAVEN**	**I-RAVEN**	**F-RAVEN**
Pixel-level similarity:			
MSE (Mean context)	6.44	14.85	14.93
RASE (c6+c8)	9.34	17.16	17.47
Best pixel-level	9.34	18.69	18.86
ImageNet-pretrained features (frozen):			
Cosine similarity (Mean context)	5.68	15.25	13.64
MAE (c6+c8)	8.46	18.76	19.68
Best frozen features	8.81	19.59	20.36
RAVEN property-adapted features:			
Cosine similarity (Mean context)	5.17	19.81	15.84
MAE (c6+c8)	7.59	19.15	18.06
Best adapted features	7.64	20.71	19.26
Random Baseline	**12.5**	**12.5**	**12.5**

Table 3.7 quantifies the practical implications through evaluation of heuristic solvers. The heuristic analysis confirms that increased discrimination margins enable similarity exploitation. While RAVEN maintains all strategies below random baseline (best: 9.34%), I-RAVEN and RAVEN-FAIR exhibit modest vulnerability to learned similarity features, with the best similarity-based strategies reaching 20.71% on I-RAVEN and 20.36% on RAVEN-FAIR.

This demonstrates a trade-off in answer set construction: RAVEN successfully mitigates similarity bias (7–9% heuristic accuracy) but suffers from severe answer set bias (context-blind models exceeding 90%). I-RAVEN eliminates answer set bias while RAVEN-FAIR substantially reduces it (around 12.5%

and 17.2% context-blind performance), but both modestly reintroduce similarity bias (20.71% and 20.36% heuristic accuracy).

The improved benchmarks represent a net gain—trading a bias exploitable above 90% for one around 20%. However, these heuristic results likely underestimate true exploitability. While simple similarity strategies achieve only modest gains above random baseline, sophisticated deep learning models could potentially achieve higher performance by learning nuanced similarity patterns. Neural networks excel at discovering subtle data correlations and features not easily detectable by humans or simple heuristics (Geirhos et al., 2020). Presenting algorithms with both context panels and answer sets during training encourages relationship-seeking between them rather than reasoning based solely on context, creating opportunities for exploitation that extend beyond what simple heuristics reveal.

3.2.4 Implications for Model Design and Research Hypotheses

The cross-benchmark bias analysis conducted in preceding sections directly corroborates hypothesis **H2** (Sec. 1.2) of this thesis, i.e., that delineating property prediction as a sub-task may help mitigate the biases present in RPM benchmarks. The findings reveal an inherent tension in answer set construction—eliminating one form of bias (answer set) can reintroduce another (similarity). Despite sophisticated mitigation strategies, all benchmarks exhibit exploitable patterns when answer panels are accessible during training, though the severity varies substantially across bias types and benchmarks.

Our property prediction approach, introduced and detailed in Chapter 4, addresses this challenge by eliminating answer panel access during training. Since both similarity bias and answer set bias require observing answer sets during learning, the staged decomposition removes the possibility of such exploitation entirely. Models must develop compositional reasoning exclusively from context panels, directly implementing **H2** by avoiding answer set exposure during the learning phase where bias patterns would otherwise be learned.

Let us conclude the considerations on dataset biases and shortcut learning with the following observations. The nature of *shortcuts* and statistical patterns in neural network reasoning requires clarification. While we emphasize bias resistance throughout this work, pattern-based heuristics are not inherently detrimental—they are how neural networks and human cognition operate efficiently.

Neural networks base decisions on regularities observed in training data, mirroring human cognitive processes. In RPM tasks, human solvers identify salient cues that guide their reasoning: noticing that the second panel contains one additional object immediately suggests investigating progressive rules on object count, rather than systematically analyzing all possible rule types. This heuristic approach—using perceptual cues to prioritize hypotheses—makes problem-solving more efficient without sacrificing correctness. Neural networks can similarly leverage statistical regularities to guide attention toward productive analytical directions.

Problems emerge when a model's decisions are driven *exclusively* by *simple* dataset regularities that bypass the intended reasoning process. We consider a model to use shortcuts when it solves tasks in an *inappropriate manner*—through brittle, low-complexity cues rather than by reasoning over the underlying structure. If a model learns that "the answer is usually in position 3" and bases decisions on this regularity without analyzing panel properties, this constitutes harmful shortcut learning. In contrast, if a model uses the observation that "panels with increasing object counts often indicate progression rules" to guide its analysis while still verifying the rule through systematic comparison, the pattern serves as helpful heuristic guidance rather than a shortcut.

In general, there is no principled way to draw a clear line between appropriate and inappropriate use of statistical regularities: those reflecting the intended task structure and those that are incidental artifacts of how the data was produced. One could argue, following the causal perspective on shortcut

learning (Geirhos et al., 2020), that a model reasons appropriately when it discovers the true *causal* regularities of the data-generating process (the underlying generative factors that actually produce the observations). For RAVEN, the causal generative factors are the components that determine panel content (arrangement, attribute values, and the rules governing their transformations across rows). Answer set statistics, by contrast, are a by-product of how incorrect panels are constructed: they correlate with the correct answer in a given dataset, but the correct answer itself is fully determined before any incorrect panel is generated. Even when the generator is fully accessible, the interaction of its components can produce emergent statistical patterns that are difficult to anticipate, as the history of bias discovery in RAVEN itself illustrates. In most practical settings, however, the data-generating process is not accessible at all, making the distinction between causal and spurious features even harder to establish from data alone. Evaluating models across benchmark variants specifically designed to target different bias types, as done throughout this thesis, is therefore a principled, if imperfect, strategy for assessing bias robustness.

To summarise the considerations on bias and statistical patterns: bias-resistant benchmarks aim to eliminate exploitable cues that make tasks solvable without reasoning—such as answer set biases or trivial similarity patterns. However, some degree of statistical regularity is inevitable in any dataset, and minor biases that provide weak auxiliary signals while still requiring considerable reasoning can function as helpful heuristic guidance, similar to how human cognition combines fast, pattern-based intuition with deliberate analytical reasoning (Kahneman, 2011).

3.3 Visual Abstract Reasoning Datasets

The development of robust abstract reasoning systems requires carefully designed benchmarks that isolate specific reasoning capabilities while controlling for confounding biases. The following subsections survey the datasets most relevant to this thesis: variants of the **RAVEN** dataset that target different bias profiles and generalization regimes, the large-scale **PGM** benchmark designed for out-of-distribution evaluation, and complementary visual reasoning benchmarks—**ARC**, **Bongard Problems**, **VAP**, and **O3**—that test distinct aspects of abstract reasoning beyond the RPM paradigm.

3.3.1 Variants of the RAVEN Dataset

Several variants of the **RAVEN** dataset have been proposed to address specific biases or test particular generalization capabilities:

- **I-RAVEN** (Hu et al., 2021): Eliminates answer distribution biases through the *Attribute Bisection Tree* algorithm, reducing context-blind performance to chance level (12.5%).
- **RAVEN-FAIR** (Benny et al., 2021): Reduces answer set bias through iterative random attribute modification, though residual bias persists—context-blind models achieve 17.2% accuracy, above the 12.5% chance level of **I-RAVEN**.
- **A-I-RAVEN (Attributeless-I-RAVEN)** (Małkiński and Mańdziuk, 2025a): Tests rule transfer through 10 generalization regimes by withholding specific rule–attribute combinations during training across five attributes (Position, Number, Type, Size, Color), where Position and Number are treated as a coupled pair.
- **I-RAVEN-Mesh** (Małkiński and Mańdziuk, 2025a): Augments **I-RAVEN** with a line-based mesh component overlaid on existing panels, requiring simultaneous reasoning over base **I-RAVEN** rules and mesh-specific Number and Position attributes, designed for transfer learning evaluation.

- **I-RAVEN-X** (Hersche et al., 2026): Extends **I-RAVEN** with a configurable number of context columns and a dynamic attribute value range, enabling *complexity scaling studies*. The dataset operates on symbolic attribute representations, assuming oracle visual perception.
- **I-RAVEN-X with Perceptual Uncertainty** (Camposampiero et al., 2025): Extends **I-RAVEN-X** by introducing *confounding attributes* and *smoothed attribute distributions* to simulate imperfect perception symbolically, revealing dramatic performance drops in Large Reasoning Models.

3.3.2 Procedurally Generated Matrices (PGM)

PGM (Barrett et al., 2018) contains 1.4 million procedurally generated RPM-style matrices designed to test *systematic generalization* through seven controlled regimes with varying train–test attribute divergence. Unlike **RAVEN**'s focus on diverse visual configurations across seven spatial arrangements, PGM employs a single arrangement equivalent to RAVEN's *distribute-nine* configuration, augmented with an additional *background layer* absent from RAVEN. This design emphasizes out-of-distribution reasoning by supporting 1–4 simultaneous rules per matrix and systematically manipulating which attribute combinations appear during training versus testing. PGM proves particularly suitable for probing whether learned representations transfer beyond specific attribute distributions, as explored in Section 6.1 of Chapter 6, where we evaluate cross-dataset generalization and visualize PGM tasks.

3.3.3 Abstraction and Reasoning Corpus (ARC)

ARC (Chollet, 2019) comprises 1,000 manually crafted grid-based transformation tasks (400 training tasks, 400 public evaluation tasks, and 200 private test tasks), designed to test *core knowledge priors* through few-shot learning, with each task providing typically 3 demonstration input–output pairs before requiring generalization to novel grids. Problems involve colored cell manipulations testing concepts like objectness, symmetry, counting, and spatial transformations—operations humans solve intuitively but that challenge current AI systems. Unlike **RAVEN**'s large-scale statistical learning regime (42,000 training samples from a total of 70,000 tasks), ARC emphasizes rapid concept acquisition from minimal data, positioning it as a complementary benchmark for sample-efficient abstract reasoning. The benchmark continues to evolve, with **ARC-AGI-2** (Chollet, 2025) providing increased task complexity. Figure 3.7 illustrates a sample task.

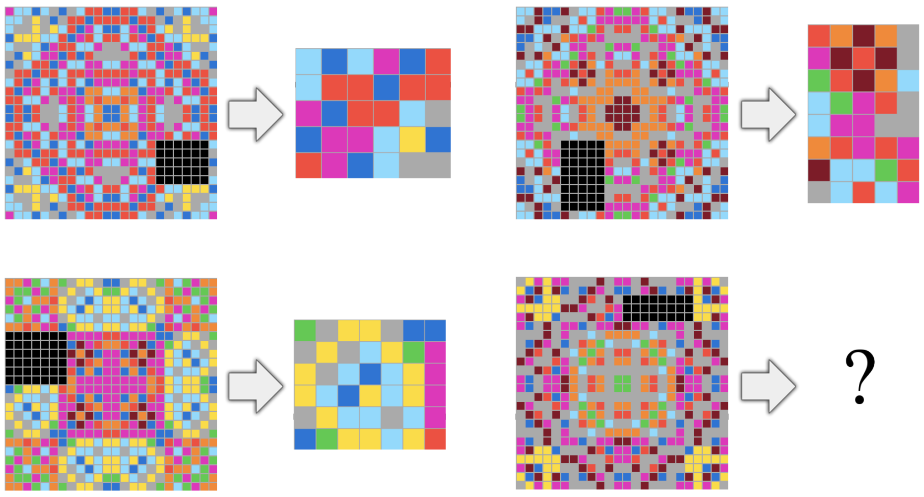


FIGURE 3.7: Sample **ARC** task (reproduced from Fig. 4 of Chollet 2019) in which the output grid contains the region that appears black (absent) in the corresponding input grid.

3.3.4 Bongard Problem Datasets

The original **Bongard Problems** (Bongard, 1970) consist of 100 manually designed visual puzzles requiring *concept induction* from positive and negative examples—six images satisfying a visual concept versus six violating it. Unlike **RAVEN**'s rule-based matrix completion, Bongard problems test pure concept learning where the discriminating feature may depend on global context, topological properties, or subtle relational patterns discoverable only through contrastive analysis. **Bongard-LOGO** (Nie et al., 2020) scales this paradigm to 12,000 procedurally generated problems across free-form shapes, geometric primitives, and abstract concepts, with programmatic annotations enabling systematic evaluation. **Bongard-HOI** (Jiang et al., 2022) extends to natural images testing human-object interaction concepts, employing compositional negative mining to ensure that discrimination requires semantic understanding rather than superficial visual cues. Figure 3.8 illustrates a sample **Bongard-LOGO** problem.

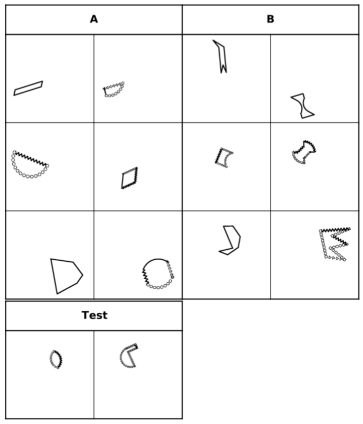


FIGURE 3.8: Sample **Bongard-LOGO** problem (reproduced from Fig. 1c of Nie et al. 2020) in which set A images contain only convex shapes, while set B images contain non-convex shapes.

3.3.5 Related Visual Reasoning Benchmarks

Several benchmarks exhibit structural similarities to RPM while introducing distinct task formulations. **Visual Analogy Problem (VAP)** (Hill et al., 2019) employs a 2×3 grid of context panels with a four-option answer set, testing analogical reasoning through row-wise pattern completion similar to RAVEN but with reduced spatial complexity. Crucially, the upper and lower rows belong to distinct visual domains, so the solver must abstract the relation expressed in the source domain and transfer it to a structurally different target domain—a requirement that distinguishes VAP from pure pattern matching and constitutes its primary difficulty. **Odd One Out (O3)** (Ruiz, 2011; Mańdziuk and Żychowski, 2019) inverts the standard RPM paradigm by requiring identification of the panel that violates shared properties among context panels, rather than completion of a missing one, as illustrated in Figure 3.9.

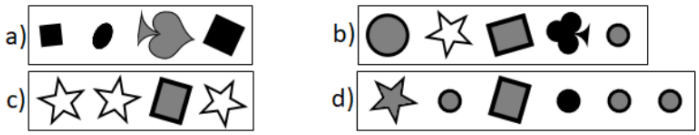


FIGURE 3.9: Sample **Odd One Out (O3)** tasks (reproduced from Fig. 10 of Mańdziuk and Żychowski 2019). Each row presents a set of panels sharing a common visual property; one panel in each row deviates from the others through a difference in shape, size, or shading.

These tasks share RAVEN's reliance on visual pattern recognition and rule-based reasoning, making them natural targets for adaptation of RPM-solving methodologies.

Beyond RPM-style matrices, several benchmarks probe complementary aspects of abstract visual reasoning. **SVRT** (Fleuret et al., 2011) provides 23 binary shape classification tasks testing topological relationships (e.g., "same shape" vs. "different shape"), serving as a minimal perceptual reasoning baseline. **CLEVR** (Johnson et al., 2017) introduces compositional visual question answering through 700,000 programmatically generated questions about synthetic 3D scenes, requiring joint language–vision processing to parse compositional queries about object attributes and relations. **GQA** (Hudson and Manning, 2019) extends this paradigm to 22 million questions over real photographs from Visual Genome, emphasizing compositional reasoning under natural image conditions. While these benchmarks test important visual reasoning capabilities, they differ from RPM tasks by incorporating linguistic queries or focusing on perceptual classification rather than rule-based pattern completion. For a broader overview of Abstract Visual Reasoning tasks, benchmarks, and emerging research directions beyond the scope of this thesis, see Małkiński and Mańdziuk (2023).

3.4 Related Work on Neural Approaches to RPM Solving

Computational approaches to abstract reasoning through visual pattern completion have a substantial history in cognitive science and artificial intelligence, with early work employing symbolic reasoning, heuristic search, and qualitative spatial representations (Evans, 1964; Lovett et al., 2007; Kunda et al., 2010; Strannegård et al., 2013). The advent of deep learning intensified this interest, with several dozen neural methods for solving RPMs emerging in recent years. For systematic reviews, we refer the reader to Małkiński and Mańdziuk (2025); here we focus on aspects particularly relevant to our approach.

Modern neural approaches commonly adopt architectures that separate visual perception from relational reasoning. A visual encoder (typically ResNet (He et al., 2016) or similar CNN architecture) extracts features from individual panels, while a reasoning module processes these representations to identify patterns and select answers. Representative examples are described below.

- **WReN** (Barrett et al., 2018) applies relation networks (Santoro et al., 2017) to score answer panels by computing pairwise relationships between context and candidate panels. The relation module explicitly models interactions between all panel pairs to identify relational patterns.
- **SRAN** (Hu et al., 2021) generates embeddings for row pairs and employs stratified reasoning to identify rules at different abstraction levels. This hierarchical approach enables the model to capture both low-level features and high-level relational patterns.
- **PoNG** (Małkiński and Mańdziuk, 2025b) employs a pathways block in the reasoner comprising four parallel pathways, where two employ novel normalized group convolution operators that split panel embeddings into groups and apply convolutions with shared weights, enabling the model to jointly reason over individual rows and pairs of rows.
- **PredRNet** (Yang et al., 2023) stacks reasoning blocks that iteratively predict and refine features, minimizing discrepancy between predicted and actual panel representations. Each iteration progressively extracts more abstract relational patterns from the visual embeddings.
- **SCAR** (Małkiński and Mańdziuk, 2024) introduces a novel **Structure-Aware dynamic Layer** (**SAL**) that adapts its weight matrices to the structure of the considered AVR problem, without making any *a priori* assumptions about the number or configuration of panels, enabling a single model to solve multiple abstract visual reasoning tasks.

While these architectures separate perception from reasoning, both components typically train simultaneously through end-to-end optimization on the answer selection objective, sometimes with auxiliary supervision such as rule classification.

Benny et al. (2021) introduced two evaluation protocols that formalize how models access answer panels during inference. In the **Multiple-Choice (MC) protocol**, the model processes all answer panels simultaneously, allowing internal comparison across candidates; notable examples include CoPINet (Zhang et al., 2019a), Rel-AIR (Spratley et al., 2020), and MXGNet (Wang et al., 2020). The **Single-Choice (SC) protocol** restricts access to one answer panel at a time, requiring the model to score each candidate independently; methods using this protocol include SCL (Wu et al., 2020), MRNet (Benny et al., 2021), and WReN (Barrett et al., 2018).

The standard SC implementation evaluates each answer panel separately by completing the matrix eight times—once with each candidate inserted into the query position. This produces eight completed matrices, each processed independently by the model. The model assigns a score to each panel estimating its fit with the pattern. A scoring module might use a linear layer or simple neural network to generate logits that are aggregated into a probability distribution and optimized using cross-entropy loss. This protocol reduces susceptibility to answer set biases because answer panels are evaluated independently, with information combined only at the loss function level.

3.4.1 Augmentation

In the RAVEN dataset, where tasks are represented as matrices of images, augmentation can be applied at three distinct levels: image-level (as in Małkiński and Mańdziuk (2022)), matrix-level (as in Kim et al. (2020)), and property-level (as in Kim et al. (2020)).

Image-level augmentation applies transformations to individual panels, including horizontal and vertical flipping, rolling, rotation, transposition, and shuffling elements within panels (Małkiński and Mańdziuk, 2022). These transformations require careful application because they may alter spatial relationships between objects, which often constitute the rules governing the matrix. Modifying object positions can violate the underlying patterns the model should learn, potentially introducing contradictory training signals.

Matrix-level augmentation operates on the arrangement of panels rather than their internal content. Kim et al. (2020) rearrange panel positions and introduce noise by replacing panels with randomly sampled panels. Matrix-level transformations face a practical constraint in classical choice-based RAVEN evaluation: the answer set panels connect directly to the query panel, so transforming the matrix can change which panel serves as the query. Kim et al. (2020) address this issue by applying matrix augmentation only to context panels within their contrastive module, ensuring transformations do not interfere with the answer selection process.

Property-level augmentation modifies attribute values while preserving structural relationships and rule compliance (Kim et al., 2020). For instance, if all figures in a matrix share the same shape, that shape can be uniformly substituted without altering the relationships between panels. This augmentation type also requires verification that property changes do not violate the governing rules—changing colors, for example, must respect any color-based patterns. Unlike matrix-level augmentation, property-level changes generally pose fewer complications with answer sets because the relationships between panels remain intact even as specific attribute values shift.

Both Małkiński and Mańdziuk (2022) and Kim et al. (2020) integrate augmentation within contrastive learning frameworks, where augmented examples serve as positive or negative pairs for metric learning.

3.4.2 Contrastive learning

Contrastive learning is a method used to craft valuable representations of input data, which are typically subsequently employed in downstream tasks. In this learning regime, the model aims to bring similar instances closer to each other in the representation space while pushing dissimilar instances farther apart. Pairs of instances are commonly utilized, where instances with similar characteristics are referred to as *positive pairs*, and those intended to be dissimilar are termed *negative pairs*. In contrastive learning, a baseline instance known as an *anchor* is often employed as a reference point for comparison with other instances. Consequently, a positive pair comprises the anchor and a positive example, while a negative pair involves the anchor and a negative example. Within this framework, a similarity measure is employed to maximize the similarity for positive pairs and minimize it for negative pairs. This approach facilitates learning task-relevant features from negative pairs by capturing distinguishing characteristics, while positive pairs help identify features of the input data that the model should disregard.

Contrastive learning is a prevalent and well-established approach for the RPM task (Zhang et al., 2019b; Hu et al., 2021; Małkiński and Mańdziuk, 2022; Kim et al., 2020). Given that in RPM problems the model has access to an answer set with eight panels and must select the correct answer from these options, a natural consideration is to employ contrastive learning, treating the correct panel as a positive instance and the wrong panels as negative instances. The RAVEN dataset (Sec. 3.1.1) appears particularly well-suited for this approach, as the creators intentionally designed the answer set to be challenging, generating incorrect panels that closely resemble the correct answer, often differing only by a single property. These challenging negative examples serve as highly efficient signals for learning crucial features from the input data. To fully implement this approach, the method needs an anchor for comparison.

In **CoPINet** (Zhang et al., 2019b), the aggregated context panel embedding serves as the anchor, which is then contrasted against each answer panel embedding individually. By contrast, the authors of the **SRAN** model (Hu et al., 2021) generate embeddings for pairs of rows. Anchor instances are formed using the first and second rows, as these sections are fully known, not masked, and remain unchanged due to matrix completion with answer panels. Positive and negative latents are derived from the average of the latent of the 1st and 3rd rows and from the latent of the 2nd and 3rd rows. Whether the matrix was completed with the correct or incorrect panel from the answer set dictates whether the entire latent serves as a positive or negative example.

Other approaches do not use an answer set at all, instead creating pairs through alternative methods that bypass answer panel selection. A popular strategy for generating positive and negative pairs involves their creation based on the applied rules rather than relying on the index of the correct answer. This perspective was evaluated in Małkiński and Mańdziuk (2022), which utilized shared rules among tasks for pair creation. Given that a task might involve the application of multiple rules, the authors adopted multilabel contrastive learning. In this methodology, positive pairs consist of tasks that share at least one rule, while negative examples are tasks that do not share rules. Furthermore, this work adheres to the standard contrastive learning protocol, which includes creating augmented views of a sample (in this case, image-level panel augmentation as described in the previous section). This process entails placing the augmented views into a batch, maximizing the similarity between views from the same source sample, and minimizing the similarity between one view and every other instance in the minibatch except the paired view. They consider this augmentation step optional, given that they already have positive and negative examples from the rule-based procedure.

As described in Section 3.4.1, augmentation is commonly integrated within contrastive learning frameworks. The authors of Kim et al. (2020), for instance, implemented matrix-level augmentation, such as replacing a panel with noise or rearranging panels within the matrix. They also applied property-level augmentation, altering the same property in all panels without violating the established rules. The au-

thors classify positive pairs as *analogical pairs* and negative pairs as *non-analogical pairs*, a distinction that, in our view, underscores the focus on task structure. They introduce two types of analogical pairs: inter-problem pairs involving property-level augmentation and intra-problem pairs involving the replacement of a panel with noise. Non-analogical pairs are generated through the rearrangement of panels within the original problem. In all these pairs, the anchor is the original problem instance from which the transformation is derived.

3.4.3 Large Language Models and Vision-Language Models for Abstract Visual Reasoning

The emergence of Large Language Models (LLMs) and Vision-Language Models (VLMs) has led to a distinct line of work on Raven-style abstract visual reasoning. Recent approaches explore whether symbolic or textual interfaces can expose RPM structure to models originally trained for language, falling into two methodological families: *text-based conversion methods* that extract ground-truth attributes from dataset metadata and format them as natural or symbolic language, and *direct visual processing* with multimodal architectures that operate on raw images.

Text-based conversion from ground-truth attributes. Gendron et al. (2023) introduce RAVEN^T, a text-based transcription of the original RAVEN dataset restricted to center-single matrices. In RAVEN^T-Text, each panel is described in natural language (e.g., "a large orange circle rotated at 90 degrees"), whereas RAVEN^T-Symbolic uses compact tuples of discrete symbols (e.g., `[(D, B, F, F,)]`) encoding shape, size, color, and orientation. They evaluate both *open-ended* variants, where an LLM generates the missing panel description (RAVEN^T-opqa), and *multiple-choice* variants (RAVEN^T-mcqa), where it selects from eight candidates. GPT-4 (Achiam et al., 2023) performs substantially better on symbolic multiple-choice prompts than on open-ended natural language, suggesting that choice-based formats simplify the task and may facilitate shortcut solutions, while open-ended generation provides a stricter probe of reasoning ability, as further discussed in Sec. 4.1. Hu et al. (2023a) extend this approach to all RAVEN arrangements through a *multi-level abstraction framework* for zero-shot reasoning. They apply *naming abstractions* that encode attributes as numerical symbols (e.g., triangle= 3, pentagon= 5), and *decomposition abstractions* that split puzzles into independent sub-prompts per attribute, reducing the cognitive load on the model. Unlike RAVEN^T's generative approach, they explicitly score candidates by computing length-normalized log-probabilities over full prompts. GPT-3 (Brown et al., 2020) achieves strong accuracy on center-single configurations, with performance increasing further under attribute-wise decomposition and reaching above 90% on distribute-nine under full decomposition. Structured symbolic factorizations enable strong zero-shot analogical reasoning when images are converted to text.

Hersche et al. (2026) investigate similar text-based conversion on I-RAVEN and introduce I-RAVEN-X, a benchmark with a configurable number of columns and attribute ranges $m \in \{50, 100, 1000\}$, to systematically compare LLMs against the neuro-symbolic ARLC model. They extract exact integer-valued attributes (shape, size, color) from dataset metadata and present them to GPT-4 and Llama-3 70B (Grattafiori et al., 2024) through *disentangled, per-attribute text prompts*—a minimal formatting variant similar to Hu et al. (2023a)'s numerical symbols. Using *predictive classification*—where the LLM outputs a single integer per attribute without seeing candidates, and a downstream step selects the best-matching answer—GPT-4 reaches strong accuracy on I-RAVEN center-single with self-consistency sampling (majority voting over multiple temperature-sampled predictions (Wang et al., 2022)). However, a per-rule breakdown exposes significant weaknesses: while GPT-4 handles constant, progression, and distribute-three rules reliably, accuracy drops sharply on *arithmetic* rules. On I-RAVEN-X with dynamic range $m = 1000$, arithmetic accuracy collapses to 8.4% for GPT-4 and 0.4% for Llama-3 70B, whereas ARLC maintains near-perfect accuracy without retraining, thanks to structured vector-symbolic represen-

tations that preserve arithmetic semantics. Performance on other rules also degrades at extended ranges, though less severely. Even with perfect ground-truth attributes, GPT-4's dramatically low arithmetic accuracy on $m = 1000$ reveals systematic failures on out-of-distribution numerical ranges—reinforcing the generalization limitations of LLMs discussed in Section 2.3.2. The authors argue that LLMs lack the structured compositional representations needed for symbolic arithmetic, whereas ARLC's vector-symbolic encoding preserves semantic invariance across dynamic ranges.

Camposampiero et al. (2025) extend I-RAVEN-X by introducing perceptual uncertainty at the attribute level: confounding attributes (rule-irrelevant distractors, e.g., texture when only shape matters) and smoothed attribute distributions (replacing exact values with probability distributions). Instead of receiving crisp numerical attributes, LLMs must reason over textual descriptions of uncertain perceptual evidence. Large Reasoning Models (LRMs) such as o3-mini (OpenAI, 2025) and DeepSeek R1 (DeepSeek-AI, 2025) experience catastrophic degradation—o3-mini drops from 86.6% to 17.0%, DeepSeek R1 from 80.6% to 23.2%—whereas ARLC remains at up to 88.0%, only modestly below its 98.6% baseline, demonstrating that robustness to uncertain perception remains a central challenge for LLM-based and LRM-based solvers.

Direct visual processing with VLMs. In contrast to text-based approaches that extract attributes from metadata, multimodal transformers operate directly on RPM images. Zhang et al. (Zhang et al., 2024) systematically evaluate GPT-4V (Achiam et al., 2023) and Gemini Pro (Gemini Team, Google, 2023) on Mensa IQ tests, IntelligenceTest, and RAVEN, feeding raw images without symbolic conversion. Reported accuracies are well below text-based methods across all benchmarks, and fine-grained analysis reveals that GPT-4V's reasoning improves substantially when given oracle textual panel descriptions, implicating perception as the primary bottleneck. However, domain-specific fine-tuning can significantly improve performance: IDEFICS-2 (8B VLM) (Laurençon et al., 2024), fine-tuned directly on RAVEN images, reaches ∼91% validation accuracy (HuggingFace M4 Team, 2024), though generalization to I-RAVEN-X's extended layouts and attribute ranges remains unexplored.

3.4.4 Neuro-Symbolic Architectures

Neuro-symbolic architectures for solving RPMs typically follow the same frontend–backend division common in deep learning approaches, but replace the neural backend with a largely symbolic reasoning module for explicit rule inference (Zhang et al., 2021a; Hersche et al., 2023b). This sharper separation between perception and reasoning yields improved interpretability and supports systematic generalization, though at the cost of substantial task-specific design: specialized symbolic operations, predetermined or learnable rule representations, and domain-tuned attribute encodings.

PrAE (Zhang et al., 2021a) introduces probabilistic abduction using an object CNN to generate attribute probability mass functions (PMFs), followed by predetermined rule templates specialized for each RAVEN rule type (*constant*, *progression*, *arithmetic*, *distribute-three*). **ALANS** (Zhang et al., 2022) eliminates manual templates by learning rule representations through trainable operator matrices grounded in abstract algebra, achieving improved generalization while requiring attribute labels for supervision. In the **NVSA** model (Hersche et al., 2023b), the frontend (*perceptor*) extracts *objects* and their attributes from each panel, converting them into a probability mass function (PMF) using a Vector Symbolic Architecture (VSA) (Kleyko et al., 2022) representation. Due to the high dimensionality of VSA, the perception module captures all objects in a single pass. Each attribute has an associated *codebook* containing vectors for all possible values of that attribute. These values are *bound* together to form a single vector representing an individual object, which can be *bundled* to represent multiple objects. By computing similarity between the ResNet-generated vector and the dictionary, a PMF over attributes is generated and passed to the reasoner.

Within the reasoner, the PMF is transformed into a new VSA representation suited for reasoning over RAVEN rules. The model determines the applied rule by performing specific operations—each rule has a custom-built function—that capture relationships between *context panels*. For example, to estimate the probability of an arithmetic rule, the system measures similarity between the representation of the third panel in a row and the binding of the first and second panels. The most probable rule generates the expected representation for the missing panel, which is converted into a PMF using associative memory search. Answer selection compares the predicted PMF against candidate PMFs using the KL-divergence. While all operations are differentiable, only the frontend is trainable; codebooks and reasoning operations remain fixed. The model is trained via supervised learning (requiring attribute labels) and end-to-end reinforcement learning with an auxiliary rule classification task. NVSA achieves 99.0% on **I-RAVEN** with attribute labels but only 88.1% end-to-end. However, NVSA relies on custom-built operations for each RAVEN rule type (*constant*, *progression*, *arithmetic*, *distribute-three*), requiring exhaustive search across all rule-attribute combinations. Subsequent work addressed this by replacing these per-rule operations with more general mechanisms.

Learn-VRF (Hersche et al., 2023a) eliminated per-rule templates by learning VSA rule formulations as soft assignment problems, achieving one-pass learning but struggling with position-dependent rules due to lack of spatial context. **ARLC** (Camposampiero et al., 2024) addressed this by incorporating context augmentation—using information from multiple panel pairs when selecting rules—and extending rule templates to accommodate more diverse rules. **Rel-SAR** (Sun et al., 2025) introduced diverse atomic vector types (Random, Numeric, Circular, Boolean) supporting distinct semantic operations, replacing per-rule templates with generic numerical and logical relation functions. Neural networks predict relation parameters (*operator powers*) rather than hardcoding per-rule operations. Achieving strong accuracy on both **RAVEN** and **I-RAVEN** via end-to-end training with rule labels only, Rel-SAR demonstrates robust perception-reasoning integration: while NVSA and ALANS suffer accuracy drops without attribute labels, Rel-SAR maintains near-identical performance with and without attribute supervision. Due to Rel-SAR training on rule structure (operator powers) derived from rule labels, the method avoids answer set bias.

3.4.5 Limitations of Neuro-Symbolic and LLM-Based Approaches

Neuro-symbolic methods demonstrate strong generalization to out-of-distribution settings such as extended attribute ranges or larger matrix constellations (Hersche et al., 2026; Camposampiero et al., 2024), but achieve this through custom-built architectures that explicitly model rules, properties, and their interactions. Despite progressive abstraction—from PrAE's per-rule templates to ARLC's learnable assignments to Rel-SAR's generic relation functions—all methods remain tailored to RAVEN's structure, and extending them to novel visual reasoning tasks requires manual redesign of representations and operations. Strong performance further depends on auxiliary supervision: neither NVSA in end-to-end mode (which requires rule classification losses alongside REINFORCE (Williams, 1992)) nor Rel-SAR (which requires rule labels to supervise operator power prediction) achieves reasoning from answer indices alone.

LLM-based approaches, by contrast, offer zero-shot reasoning without task-specific construction, but with mixed results. Even when given perfect ground-truth symbolic attributes, they show systematic weaknesses on arithmetic rules in some settings, and under extended attribute ranges performance degrades substantially across all rule types (Hersche et al., 2026); performance also degrades sharply under perceptual uncertainty (Camposampiero et al., 2025). The two paradigms thus present a complementary trade-off: neuro-symbolic models generalize reliably within RAVEN's structure but require explicit task engineering, while LLMs transfer broadly but remain fragile to out-of-distribution inputs.

Chapter 4

Abstract Compositional Transformer

Building upon the dataset biases and evaluation challenges identified in Chapter 3, this chapter presents our novel staged self-supervision framework that addresses these limitations through *property prediction* rather than direct choice selection. The **Abstract Compositional Transformer (ACT)** represents the first stage of our two-phase methodology, designed to learn representations of abstract reasoning patterns while avoiding the statistical shortcuts that plague existing approaches. This staged decomposition—predicting panel properties before making answer choices—directly implements **H1** (Sec. 1.2): decomposing reasoning tasks into distinct stages facilitates obtaining better solvers.

This chapter demonstrates how decomposing the classical RPM task into property prediction and choice making (detailed in Chapter 5) enables more robust and interpretable reasoning systems. We begin by developing the conceptual framework for this approach, then present the ACT architecture, training methodology, and comprehensive experimental evaluation.

4.1 Motivation: Self-Supervised Property Prediction over Choice-Based Evaluation

The foundations presented in Chapter 2 highlight the propensity of deep neural networks to discover **statistical shortcuts**: seeking the most computationally efficient paths to achieve favorable values of the loss function. When a more straightforward approach exists for attaining high performance, models inevitably discover it, regardless of whether it aligns with the intended reasoning methodology or enables generalization to similar tasks.

As demonstrated in Chapter 3, **RPM** datasets, particularly **RAVEN**, exhibit significant biases (Sec. 3.2.2). The restricted output space of only eight possible answers renders the task particularly susceptible to biases. Rather than engaging in the intended reasoning process—identifying *context panel* properties, inferring patterns and relationships, and applying those patterns to predict the final answer—

the task is reducible to statistical discrimination among a fixed candidate set. Because the correct answer is present in the input by definition, the candidate set introduces distributional information that models may exploit without engaging with the underlying task structure (Sec. 3.2.2). More directly, the answer panels themselves provide substantial information about the query panel: all property values of the query panel are embedded within the answer set, enabling identification without rule inference.

Tasks framed as property construction remove this vulnerability entirely—because no candidate answers are provided, shortcuts based on answer distributions become unavailable. As established in our analysis of shortcut learning (Chapter 2), discriminative models tend to identify features enabling class distinction without necessarily understanding the underlying objects. A model predicting complete symbolic property vectors, by contrast, faces a substantially larger and more structured output space, which raises the bar for shortcut exploitation and encourages the development of more informative internal representations. Other shortcut pathways may still exist, but shortcuts that rely on answer set distributions are structurally eliminated.

The open-ended nature of construction tasks also benefits learning through inherent ambiguity: when multiple property assignments are equally valid, models must develop deep task understanding rather than memorizing a single correct answer, and methods that reward all valid solutions equally (Papineni et al., 2002; Zhu et al., 2017) prove superior to binary-reward approaches. This aligns more closely with real-world problem-solving, where ambiguity is abundant and problems often admit multiple valid solutions. By framing the task as property construction, we emphasize the approach's generalizability beyond **RAVEN** to other abstract reasoning benchmarks that require actual understanding rather than pattern matching.

Beyond bias resistance and task difficulty, construction-based tasks yield models whose intermediate outputs are directly interpretable. Because the model must produce an explicit symbolic description of the query panel before any choice is made, it is possible to inspect exactly which properties were predicted and where errors occur—a transparency unavailable in monolithic approaches that map directly from image to answer index.

While property prediction constitutes the main contribution of this chapter, the representations learned by *ACT* are subsequently used for answer selection, enabling comparison with existing methods through established evaluation protocols. The choice-making stage is described in Chapter 5.

4.2 Abstract Compositional Transformer Architecture

The Abstract Compositional Transformer represents our novel architecture designed specifically for property prediction in abstract reasoning tasks. This section details the model's core components and their integration within the staged self-supervision framework.

4.2.1 Architecture Overview

ACT is a deep learning model that takes as input an RPM puzzle and maps it to **property vectors**—symbolic descriptions of the contents of each panel (Sec. 4.2.5). At a high level, the model consists of three components (Fig. 4.1): an **Image Tokenizer**, a **Transformer**, and a **Property Predictor**. The image tokenizer converts visual panels into sequences of tokens suitable for transformer processing. The transformer processes these token sequences to learn representations of abstract reasoning patterns. Finally, the property predictor maps transformer outputs to property vectors for each panel. During training, one panel is replaced by a learnable mask before tokenization; because the model predicts symbolic descriptions for all nine panels, it must use the remaining context to infer the properties of the masked one (Sec. 4.3).

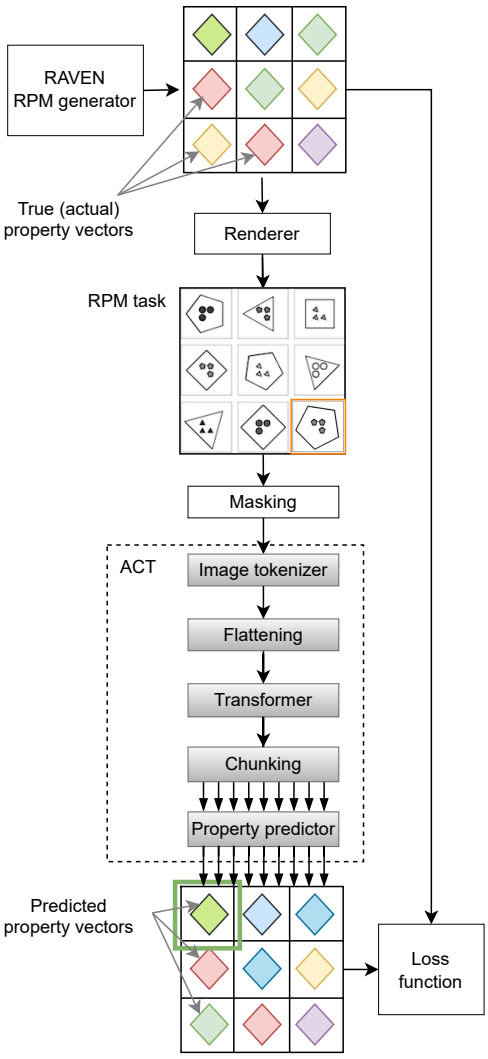


FIGURE 4.1: Architecture of ACT and its learning from completed RPM tasks, with one of the panels (context or query panel) masked out. The loss function compares the predicted and actual properties of RPM panels.

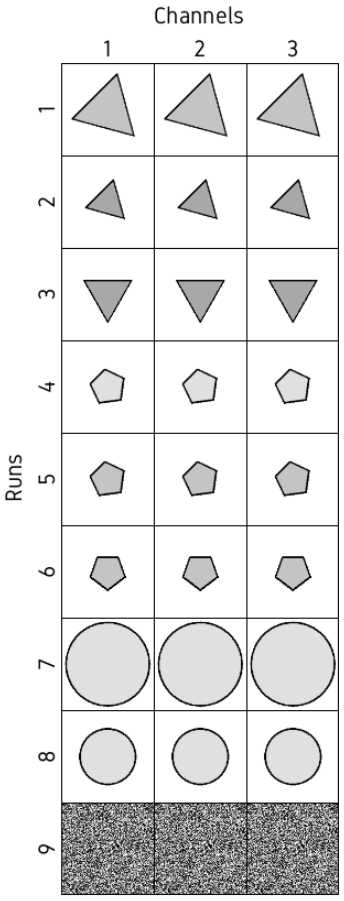


FIGURE 4.2: Extractor input under the Panel tokenizer. The extractor runs independently on each of the nine panels (rows); channels carry three copies of the single-panel image.

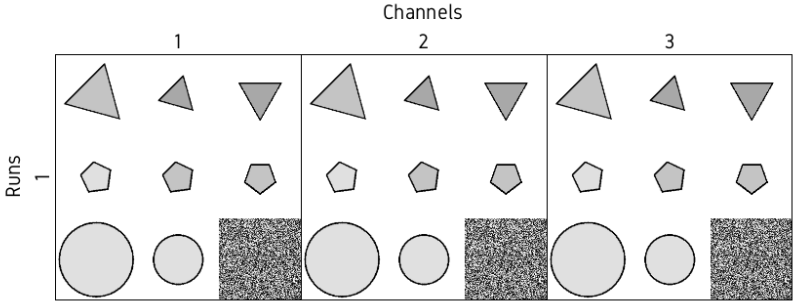


FIGURE 4.3: Extractor input under the Task tokenizer. The extractor runs once on the full task image; channels carry three copies of the combined task raster.

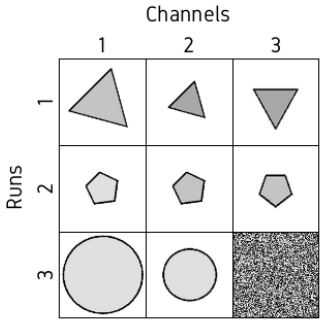


FIGURE 4.4: Extractor input under the Row tokenizer. The extractor runs once per row; the left, central, and right panels map to the channels, respectively.

4.2.2 Image Tokenization

The cornerstone of modern transformer-based models is effective tokenization— the process of converting input data into discrete tokens that capture relevant information while enabling efficient processing (Vaswani et al., 2017). Different tokenization strategies suit different tasks: NLP tokenizers that segment text into words or subwords excel at semantic tasks but prove suboptimal when character-level manipulation is required (Sennrich et al., 2016). Drawing parallels to language models, we refer to our image processing component as the **Image Tokenizer**. The tokenizer takes nine raster images of *context panels* and outputs a sequence of *tokens* that the transformer can process.

The original panel size in the RAVEN dataset is 160×160 pixels, which we downsample to 84×84 pixels. Preliminary experiments showed that this resolution strikes an optimal balance between model performance and computational cost: the model can still reliably learn the properties of even the smallest objects, while training remains tractable within a reasonable timeframe.

Next, the nine 84×84 pixel images are transformed into a format that the *extractor* can process. The extractor—a **Convolutional Neural Network (CNN)**—transforms images into a lower-dimensional representation. Given that the extractor is a CNN, its output is a 3D tensor (width × height × channels). The image tokenizer treats each element of this tensor (a vector of channel values) as a token, reshaping the multidimensional output into a sequence of tokens.

We initialize the CNN with weights pre-trained on RGB images for all three tokenizer variants. Although this could be considered semantically questionable—since models are typically trained to extract features from authentic 3-channel RGB images—this practice is well established in deep learning and has proven successful even across domains that differ radically from natural photographs. For instance, Yang et al. (2019) proposes repurposing RGB convolutional layers of 2D models pre-trained on generic photographs for the segmentation of 3D volumetric CT scans, and Rezende et al. (2017) successfully transfers ResNet-50 weights from ImageNet to malware binary classification, where inputs are grayscale byteplot visualizations of executable code. The CNN is subsequently fine-tuned end-to-end alongside the full model, allowing it to adapt to the characteristics of RPM.

We propose three tokenization strategies that are sufficiently distinct to evaluate the model across a broad spectrum of approaches. The input format for each strategy is illustrated in Figs. 4.2–4.4, where the masked panel is also visualized; the following subsections describe each strategy in turn.

Panel Tokenizer

This approach analyzes each panel individually by independently applying the feature extractor to it (Fig. 4.2). The raster image representing each panel (8484 pixels) is processed by a CNN to produce a 3×3 array of 128-channel feature vectors, which is then flattened row-wise into a sequence of nine 128-dimensional tokens.

This process is repeated for all nine panels of the puzzle, and the resulting sequences are concatenated, producing 81 tokens in total. This tokenizer is particularly well-suited for encouraging the model to decompose the task into subproblems, as it provides information from each panel in isolation, allowing the transformer to focus on individual panel properties while learning pattern matching through attention.

Task Tokenizer

In contrast, this variant processes the entire RPM task's raster image through a single CNN invocation (Fig. 4.3). For RAVEN, this involves applying the CNN to a 252×252 pixel image (after rescaling each panel from the original 160×160 to 84×84, Sec. 4.4), resulting in an 8×8 array of 128-channel feature vectors, flattened row-wise to create a sequence of 64 128-dimensional tokens.

This approach enables analysis of relationships between all panels from the early processing stage, allowing the model to begin reasoning about cross-panel patterns already in the first convolutional layers. This provides greater compositional flexibility in task decomposition and pattern recognition.

Row Tokenizer

In this configuration, individual panels within each row are represented by stacking rasters channel-wise (Fig. 4.4). This results in three 3-channel 84×84 images corresponding to the top, middle, and bottom rows of the puzzle. The CNN is then applied independently to each of these images, producing a 3×3 array of tokens for each row. These arrays are flattened row-wise, generating nine 128-dimensional tokens. Finally, the subsequences for the top, middle, and bottom RPM rows are concatenated, resulting in a total of 27 tokens.

By stacking panel images in this manner, we directly juxtapose them in input channels, enabling the CNN to capture *low-level features* that highlight differences between individual images. The RPM images from the left, central, and right columns are mapped to individual channels, respectively; in RPM, this mapping is intuitive, potentially aiding the model in discerning differences between spatially aligned structures in panels.

Receptive Fields and Token Correspondence

The relatively small sizes of the rasters, combined with 18 convolutional layers of the CNN (cf. Table 1 in (Tan and Le, 2019)), cause the receptive fields of units in the last layer to span the entire input image. Therefore, for all tokenizers, each token may in principle depend on the *entire input raster* (a panel raster for Panel and Row tokenizers, and a task raster for the Task tokenizer). Additionally, only the Panel tokenizer is guaranteed to ensure some degree of selective correspondence between RPM panels and tokens. In the Task tokenizer, the representation is more entangled, as the receptive fields of the CNN are allowed to span multiple neighboring panels. In the Row tokenizer, the consecutive groups of nine tokens form an entangled representation of the top, middle, and bottom rows of panels. However, the degree of entanglement depends on the characteristics of features acquired in training, and the actual *effective receptive fields* can be smaller (see, e.g., (Luo et al., 2017)).

4.2.3 Transformer Component

Solving an RPM puzzle requires capturing and extrapolating the patterns of shapes, arrangements, sizes, colors, and numbers of figures across the presented panels (Fig. 1.1). This makes them an excellent match for transformer-like architectures (Vaswani et al., 2017), which have been designed to model co-dependencies between tokens in sequences. This convergence served as the primary motivation for designing the Abstract Compositional Transformer.

The transformer plays a central role in our model and is tasked primarily with analyzing the relationships between tokens and generating representations of the panels. The transformer processes one-dimensional sequences of tokens X generated by an image tokenizer. Initially, each token is encoded independently using the encoder:

$$Z = \text{map}(\text{Encoder}_{\theta_E}, X), \tag{4.1}$$

which is implemented as a dense linear layer, primarily designed to match the dimensionality of the tokens to the input dimensionality of the transformer. Subsequently, the transformer maps the sequence of encoded tokens Z to a sequence of output tokens O of the same length:

$$O = \text{Transformer}(Z;\, \theta_T). \tag{4.2}$$

The model is equipped with a *learnable positional encoding*, applied to the input tokens in Z. Since the number of tokens is constant, we encode the *absolute positions* of tokens in Z using a fixed-size learnable embedding—one entry per position, yielding 81, 64, and 27 entries for the Panel, Task, and Row tokenizers, respectively. The embedding vectors are added to respective tokens in Z before passing them to the transformer.

Internally, the transformer is a stack of transformer blocks, each consisting of a *multi-head attention mechanism* $\text{Attn}(\theta_A)$, normalization layers LayerNorm, a feed-forward network $f(\theta_f)$, and skip connections. The processing for the i-th token can be summarized as:

$$\begin{aligned} a_i &= \text{Attn}(\text{LayerNorm}(z_i;\, \theta_N);\, \theta_A) \\ m_i &= a_i + z_i \\ o_i &= f(\text{LayerNorm}(m_i;\, \theta_{N'});\, \theta_f) + m_i \end{aligned} \tag{4.3}$$

The additions $a_i + z_i$ and $f(\cdot) + m_i$ are residual connections, which improve gradient propagation during training (He et al., 2016). This is not the original Transformer architecture (Vaswani et al., 2017) but rather the BERT architecture (Devlin et al., 2018); for a detailed description of transformers and multi-head attention, see these cited works.

The transformer employs four blocks with eight attention heads, 128-dimensional token representations, and 512-dimensional feed-forward inner layers, using *GELU* activation functions, a dropout rate of 0.1, and layer normalization with $\epsilon = 0.001$.

4.2.4 Property Predictor

The property predictor maps transformer output tokens to symbolic property representations for all nine panels. The output token sequence is partitioned into nine chunks, where each chunk is a concatenation of tokens corresponding to a single panel. The predictor is applied to each chunk independently, producing nine property vectors in total. In our configuration, the predictor consists of a simple *fully connected layer* that outputs logits, subsequently used to compute the loss against the target property vector.

More formally, the mapping proceeds through a three-step process:

1. The token sequence is divided into 9 chunks of equal length.
2. Tokens within each chunk are concatenated to form a single vector.
3. Each vector is independently mapped to a property vector using a dense subnetwork.

For the Panel tokenizer (81 tokens), each chunk contains 9 tokens; for the Row tokenizer (27 tokens), each chunk contains 3 tokens; for the Task tokenizer (64 tokens), each chunk comprises 7 tokens (with the final token discarded). The nine resulting property vectors correspond to RPM panels traversed row-wise. This design requires the transformer to both combine and disentangle information from input tokens—combining information to detect cross-panel patterns while disentangling representations to produce panel-specific property vectors.

The nine property vectors produced by the predictor constitute the model's complete output—one symbolic description per panel. The following subsection defines their structure in full.

4.2.5 Property Vector Representation

As described in Sec. 3.1, the representation for a panel is a hierarchical structure that describes every element within the panel. To address the property prediction task, we developed a **property vector** representation that describes panel contents using a flat vector encompassing all panel properties. This

representation captures every possible panel in the dataset while maintaining compatibility with neural network processing requirements.

Since neural networks operate most effectively with constant-size tensors, we designed property vectors with constant dimensionality for every panel in the dataset, regardless of how many objects are present. The vector encodes: the *arrangement* (the spatial configuration of objects within a panel, as introduced in Sec. 3.1.1); the *slots* occupied by figures in each arrangement (where a slot is a spatial position within a panel that may or may not contain an object, as defined by the arrangement type); and the properties of each figure. Since we consider all possible slots across arrangements, the number of slot values corresponds to the sum of the maximum number of slots across all arrangement types, totaling 25 (see Sec. 3.1.1 for the per-arrangement breakdown).

As noted in Sec. 3.1.2, no rules are applied to figure rotation—it is randomly sampled and therefore uninformative. We omit rotation from the representation; each figure is thus described by three values: type, color, and size, adding 75 dimensions to represent the attributes of all potential figures.

In summary, the property vector comprises *101 scalar values*:

- **Arrangement**: one value encoding the arrangement type (the spatial configuration; see Sec. 3.1.1).
- **Slot occupancy**: 25 binary indicators for object presence or absence at each possible position.
- **Object properties**: 75 values representing the attributes (type, color, size) for each potential object.

4.3 Self-Supervised Training Methodology

In standard RPM evaluation, a model is given eight context panels and must select the correct answer for the ninth—the *query panel*. Rather than training directly on this choice task, ACT learns through *self-supervision*: it is trained to predict the symbolic properties of panels from the puzzle image, without requiring labeled answer choices.

To create a self-supervised training signal, we follow the paradigm of Masked Language Models (Devlin et al., 2018). However, unlike standard MLM, which substitutes individual tokens with a `[MASK]` symbol, we apply masking at the *panel level* before image tokenization—the entire masked panel raster is replaced by a **learnable mask** (Sec. 4.9.2) before the CNN processes it, so the transformer never receives explicit mask tokens but instead processes features derived from the masked image.

In addition to predicting the masked panel's properties, the model is trained to classify the properties of the remaining observed panels simultaneously. This yields two concurrent training objectives:

- **Classification**: predicting properties of the observed *non-masked* panels.
- **Prediction**: predicting properties of the *masked* panel.

Classifying the properties of observed panels provides a strong auxiliary training signal for extracting relevant visual features—features that are considerably easier to learn than those of the masked panel, which is particularly valuable at the beginning of training. This approach functions as a *complementary task*: the classification component helps create robust panel representations, while the prediction component develops the model's *pattern recognition capabilities*, which in turn enhances property identification across all panels.

4.3.1 Masking Strategies

Separating property prediction from choice making means masking is not restricted to the query panel: we additionally train the model by masking context panels, compelling it to reason about patterns from

multiple positions within the matrix rather than always extrapolating from a fixed direction. We propose two masking strategies that differ in which panel is selected for masking during training. Both strategies apply masking at the panel level, before image tokenization: the selected panel's raster is replaced by the learnable mask, and the tokenizer then processes this modified puzzle as usual.

Random Masking

In RPM, patterns can be learned from any position within the matrix, as the underlying rules (e.g., progression of object count) apply bidirectionally across panels. *Random masking* involves selecting any of the nine panels uniformly at random and masking it during training, facilitating pattern learning regardless of panel position.

Each puzzle is completed with the correct answer panel, and a randomly selected panel is masked out on the fly, ensuring that the same task has different panels masked across training epochs. The model must learn to predict properties of panels in any position—first, middle, or last in rows and columns—requiring it to master both interpolation and extrapolation of patterns present in panels.

Random masking compels the transformer to detect and reason about patterns across the entire puzzle structure. When the model must predict any panel from the remaining eight, it cannot rely on position-specific heuristics but must discover the underlying rules governing the matrix and apply them flexibly. This acts as both dataset expansion—each task generates nine training examples instead of one—and regularization, as the model learns to ground predictions in multiple contexts rather than overfitting to position-specific patterns.

Query Masking

The *query panel* refers to the final panel (bottom-right position) in an RPM task, which is typically empty and represents the target for completion. *Query masking* presents RPM tasks in their original form with only the query panel masked.

In this configuration, the model predicts the properties of the query panel while having access to all eight context panels. Since the query panel is positioned at the terminus of the third row and column, predicting its properties requires extrapolation of the properties observed in the first and second rows and columns. This mirrors the standard test-time evaluation scenario, where the objective is to select the correct panel for the query position.

Query masking enables the model to focus specifically on the query panel prediction task, making it suitable for fine-tuning after broader pattern learning. However, training exclusively with query masking limits the model's exposure to the full range of pattern recognition challenges present in the matrix structure.

4.3.2 Relevance Masking

In our implementation, we augment the property vector with global information about the task to provide additional context. Specifically, we incorporate details about the applied rules, uniformity constraints, and the number of figures in each panel. While these values are not directly utilized within the model, they play a crucial role in both the training and evaluation processes by enabling the construction of **relevance masks**.

Given that the model generates representations applicable to panels with varying arrangements and different numbers of objects, the relevance mask allows us to focus training and evaluation on relevant properties in a given context while zeroing out irrelevant dimensions. Additionally, the dataset exhibits

noise through property resampling in specific scenarios, where properties are resampled under certain conditions and subsequently marked as irrelevant.

In any given RPM task, only a subset of the 101 properties in the property vector is meaningful. For instance, a panel with a *center_single* arrangement occupies only one slot, making the remaining 24 slot indicators and their associated object attributes irrelevant. Additionally, the dataset applies uniformity resampling under certain conditions, marking affected properties as irrelevant. The **relevance mask** prevents the model from being penalized for not predicting such properties, focusing both training and evaluation exclusively on the meaningful, deterministic subset.

Relevance is determined by two complementary sources:

1. **Task-relevant properties**: determined by the task structure through a two-stage process:
   - The arrangement type determines which slot positions are relevant.
   - Each occupied slot position determines which object attributes (color, size, type) are relevant.
2. **Uniformity-relevant properties**: when uniformity is not enforced and all other requirements are satisfied, a property is resampled and marked as irrelevant (for details on uniformity computation, see Sec. 3.1.4).

A concrete illustration of how these two relevance types interact during metric computation is provided in Sec. 4.4.3 (Table 4.1).

Importantly, the model does not predict relevance directly, as this would require predicting uniformity and relational rules explicitly—an assumption that contradicts the task's foundational premise. Instead, the model predicts arrangement and slot occupancy, from which the complete set of task-relevant properties can be reconstructed. This ensures that property prediction focuses exclusively on meaningful, deterministic relationships rather than random resampled variations.

4.3.3 Loss Function

To compute the training loss, property vectors (Sec. 4.2.5) are encoded using **one-hot encoding**, which allows the model's output to be interpreted as a probability distribution over discrete property values; categorical cross-entropy then measures the divergence between the predicted distribution and the one-hot target. The one-hot encoded components comprise:

- **Arrangement encoding**: 7 values for 7 possible arrangements.
- **Slot occupancy**: 25 binary values for slot presence or absence.
- **Object attributes**: 525 values representing all possible categories for color (10), size (6), and type (5) per figure: $25 \times (10 + 6 + 5) = 525$ dimensions.

Together, these components constitute a 557-dimensional encoded representation. **Softmax activation** is applied to categorical variables and **sigmoid activation** to binary slot indicators, ensuring all outputs lie in $[0, 1]$. The cross-entropy for each variable is multiplied by an empirically tuned weight: 1.0 for arrangement, 2.83 for presence_i, and 0.85, 1.10, and 1.22 for color, size, and type, respectively—set so that each property type contributes approximately equally to the total loss. These weights were derived from one of the earliest trained models and kept fixed across all subsequent experiments (see Appendix A.1.4). The loss is computed exclusively over *relevant properties*, with relevance determined from the target property vector (Sec. 4.3.2).

4.4 Experimental Setup

In this section, we describe the datasets, training configurations, hyperparameters, and evaluation protocols used throughout our experiments. The property prediction results are presented in this chapter, while Chapter 5 extends these trained models to solve choice tasks. The Appendix provides detailed implementation descriptions, comprehensive hyperparameter optimization analysis, and additional experiments.

4.4.1 Datasets and Data Sources

The complete RAVEN benchmark (Zhang et al., 2019a) (Sec. 3.1.1), comprising training, validation, and test sets, was downloaded from the authors' website. The dataset contains *70,000 tasks* spanning seven spatial arrangements of objects, with 10,000 tasks per arrangement.

For the **I-RAVEN** benchmark (Hu et al., 2021) (Sec. 3.2.3), we generated the 14,000 tasks forming the test set using the authors' generator with default parameter settings. We did not require the training portion, as our models were trained exclusively on RAVEN data to evaluate cross-dataset generalization.

For **RAVEN-FAIR** (Benny et al., 2021) (Sec. 3.2.3), we generated the test set using the authors' implementation. Similarly, we trained exclusively on RAVEN data.

4.4.2 Training Configuration

We trained nine model configurations in total, combining three tokenization strategies (Panel, Task, Row) with three training strategies. Each training strategy defines which masking mode or sequence of masking modes is applied across the full training schedule. Training utilized the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.001 and a batch size of 64.

- **Random masking strategy**: The random masking mode (Sec. 4.3.1) is applied exclusively throughout training (200 epochs). In each training step, a randomly selected panel (one of 9) is masked out on the fly to ensure varied masking across epochs.
- **Query masking strategy**: The query masking mode (Sec. 4.3.1) is applied exclusively throughout training (200 epochs). RPM tasks are presented in their original form with only the query panel masked; non-masked panels are predicted as well.
- **Combined masking strategy**: Training is partitioned into two phases. Phase 1 applies the random masking mode (Sec. 4.3.1) for 200 epochs as the main learning stage. Phase 2 applies the query masking mode (Sec. 4.3.1) for up to 30 epochs as fine-tuning; the loss weight for the query panel is multiplied by 0.01, and the loss is not applied to non-masked panels (i.e., the model is not penalized for predictions on context panels).

In all strategies, non-masked panels are predicted alongside the masked panel (Sec. 4.3.1); a loss weight multiplier of 2 is applied to the masked panel throughout, except during Phase 2 of the combined masking strategy, where this weight is reduced to 0.01. Validation is performed after each epoch; the checkpoint achieving the lowest validation error is selected for the next phase or final evaluation on the test set.

The random masking strategy facilitates learning patterns bidirectionally across the matrix and provides implicit regularization through diverse prediction contexts. The query masking strategy focuses specifically on the test-time scenario. The combined masking strategy leverages the strengths of both: broad pattern learning through the random masking mode in Phase 1, followed by task-specific fine-tuning through the query masking mode in Phase 2.

Complete architecture specifications—including detailed transformer configurations (Table A.1), tokenizer CNN architectures, predictor layer dimensions, tensor flow diagrams, and computational resource requirements—are provided in Appendix A.1.

4.4.3 Evaluation Metrics

To assess the models' capacity to predict panel properties, we define a set of test-set metrics calculated on the relevant properties only, with the target property vector acting as the source of relevance (Secs. 4.2.5 and 4.3.2).

The metrics are defined as follows:

- **Average Hamming distance (AvgH)**: The average Hamming distance between the predictions and the target on the relevant properties, averaged over tasks. For a given task, the best attainable value is 0, while the worst corresponds to the scenario with 9 objects (*distribute-nine* arrangement) and amounts to 37 (1 for the incorrect arrangement identifier, plus 9 for the incorrect presence/absence of the 9 slots, plus $9 \times 3 = 27$ incorrect values of the color, size, and type of objects).
- **Property Rate (PropRate)**: The fraction of correctly predicted relevant properties across all test tasks, computed as a global ratio rather than an average of per-task values.
- **Average Property (AvgProp)**: The fraction of relevant properties correctly predicted, averaged over individual tasks. For a given task, it amounts to 1 if all relevant properties have been predicted correctly, and 0 if none. Because the number of relevant properties varies by task, AvgProp is not equivalent to PropRate: each task contributes equally to AvgProp but proportionally to PropRate, yielding AvgProp = 0.80 versus PropRate = 0.81 (Table 4.1).
- **Correct (Corr)**: The primary metric used in subsequent evaluation. It amounts to 1 if all relevant properties of the query panel have been correctly predicted; otherwise 0. Averaged across all tasks in the test set. Corr is a strict metric: task 7 has only one wrong prediction out of nine relevant properties, yet still counts as fully incorrect. For arrangements with more figures, such as *distribute-nine* in the full RAVEN dataset, the metric becomes increasingly stringent, as all relevant properties across all figures must be predicted correctly for a task to count as solved.

Table 4.1 illustrates the metric computation on a simplified synthetic example. The example uses only two arrangement types rather than the full seven present in RAVEN: Arrangement 1 has one slot (one figure), corresponding to *center-single*; Arrangement 2 has two slots (up to two figures). Arrangement 2 does not have a direct equivalent in RAVEN: it resembles *distribute-four* and *distribute-nine* in having a single layout whose figures are optional, but with a maximum of two figures rather than four or nine. It differs from the two-figure arrangements *left-center-single-right-center-single*, *up-center-single-down-center-single*, *in-center-single-out-center-single*, and *in-distribute-four-out-center-single*, all of which contain two layouts each requiring at least one figure, whereas Arrangement 2 has a single layout and may hold one or two figures. Task 8 illustrates this: the second slot is occupied while the first is empty, so Figure 1 attributes are irrelevant and marked –.

Relevance follows a two-stage cascade (Sec. 4.3.2): arrangement determines which slots are active, and each active slot determines which figure attributes are evaluated. In tasks 3 and 6, the color and size attributes, respectively, are additionally marked as irrelevant due to uniformity resampling, which applies per attribute across all figures simultaneously; hence in task 6 both S1 and S2 are excluded.

TABLE 4.1: Simplified synthetic examples demonstrating metric computation for property prediction under two arrangement types. Arrangement 1 has one slot (one figure); Arrangement 2 has two slots (up to two figures). Relevance is determined by the ground-truth property vector: a shaded cell marks a relevant, evaluated prediction (× = incorrect, ✓ = correct); a dash (–) marks a property excluded from evaluation, either because the active arrangement does not use that slot or figure, or because the property was resampled due to non-enforced uniformity (Sec. 4.3.2). Resampling applies per attribute: in task 3, Color is excluded; in task 6, Size is excluded for both figures simultaneously. **Abbreviations:** Arr = Arrangement, Sl = Slot, T = Type, S = Size, C = Color.

ID	Arr	Arrangement 1				Arrangement 2								Corr	AvgH	AvgProp	PropRate
		Sl1	T	S	C	Sl1	Sl2	T1	S1	C1	T2	S2	C2				
1	×	×	×	✓	✓	–	–	–	–	–	–	–	–	×	3	0.40	–
2	✓	×	✓	✓	✓	–	–	–	–	–	–	–	–	×	1	0.80	–
3	✓	✓	×	✓	–	–	–	–	–	–	–	–	–	×	1	0.75	–
4	✓	✓	✓	✓	✓	–	–	–	–	–	–	–	–	✓	0	1.00	–
5	✓	–	–	–	–	×	✓	✓	✓	✓	–	–	–	×	1	0.83	–
6	✓	–	–	–	–	✓	✓	✓	–	✓	×	–	×	×	2	0.71	–
7	×	–	–	–	–	✓	✓	✓	✓	✓	✓	✓	✓	×	1	0.89	–
8	✓	–	–	–	–	✓	✓	–	–	–	✓	✓	✓	✓	0	1.00	–
Average over all tasks (AvgH, AvgProp, Corr) / global ratio (PropRate)														**0.25**	**1.13**	**0.80**	**0.81**

4.5 Evaluation

Table 4.2 presents property prediction performance across all nine model configurations (three tokenizers × three masking strategies) on RAVEN. The results reveal three consistent patterns. First, masking strategy has the largest impact: Random and Combined masking considerably outperform Query-only masking across all tokenizers. Second, tokenization strategy drives a marked divergence between classification and prediction accuracy: the Row tokenizer achieves the highest prediction accuracy (77.58% Correct), while the Panel tokenizer excels at classification (98.28%) but underperforms markedly on prediction (22.17%), reflecting where in the architecture cross-panel comparisons are made. Third, these patterns hold consistently across RAVEN, I-RAVEN, and RAVEN-FAIR, demonstrating that our approach learns genuine reasoning capabilities rather than dataset-specific shortcuts. We structure our analysis to explain these findings: we begin with masking strategy, which shows the largest performance differences, then examine tokenizer architectures, and finally validate generalization across benchmarks.

TABLE 4.2: Performance comparison across tokenizer and masking configurations on RAVEN.

Tokenizer	Masking	Property prediction				Classification	
		Correct	PropRate	AvgProp	AvgH	Correct	AvgProp
Panel	Query	1.53	62.23	58.08	4.55	97.54	99.50
	Random	20.82	82.52	80.34	2.23	98.38	99.44
	Combined	22.17	83.33	81.29	2.14	98.28	99.49
Task	Query	18.91	81.23	79.25	2.41	89.69	98.35
	Random	72.63	95.80	95.25	0.65	88.44	98.00
	Combined	75.63	96.15	95.64	0.61	88.33	97.98
Row	Query	20.56	79.96	78.15	2.66	84.03	97.68
	Random	75.44	96.17	95.69	0.60	87.64	98.22
	Combined	77.58	96.47	95.99	0.56	87.85	98.25

4.5.1 Impact of Masking Strategy

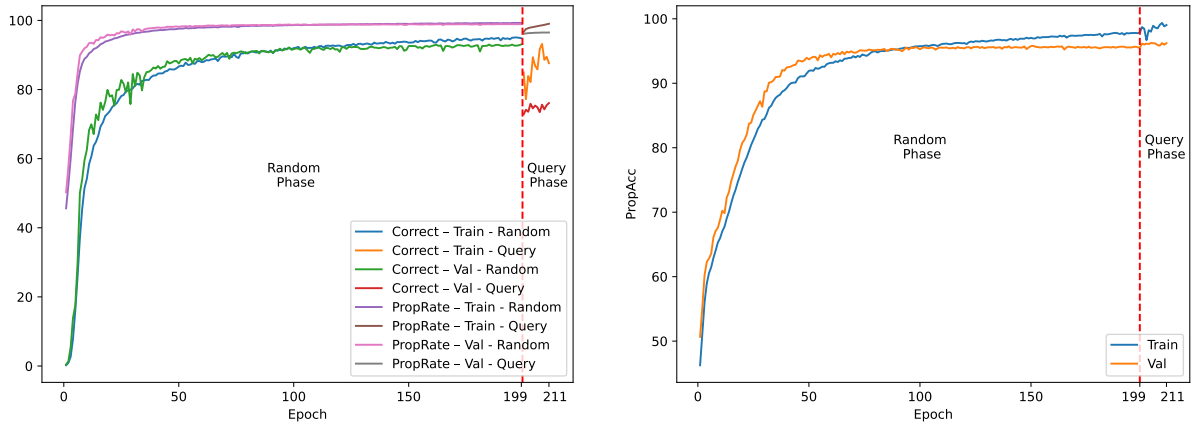
Table 4.2 reveals that masking strategy has the largest impact on property prediction performance. Across all tokenizers, Random masking markedly outperforms Query-only masking: for the Task tokenizer, performance jumps from 18.91% to 72.63% Correct. Combined masking (Random followed by Query fine-tuning) provides additional modest gains of 2–4 percentage points.

This finding appears paradoxical. Query-only masking trains the model on exactly what it will be tested on: predicting the bottom-right panel through extrapolation along rows and columns. Random masking introduces harder challenges—predicting middle panels requires interpolation, while predicting first panels requires backward extrapolation. Yet models trained on these harder, more varied prediction tasks perform better on the standard query panel prediction.

The explanation lies in how random masking forces the model to learn: when predicting any of the nine panels, the model cannot rely on position-specific heuristics but must discover the underlying rules and apply them flexibly. This confirms the dual role described in Sec. 4.3.1—dataset expansion and regularization against position-specific overfitting—and directly validates **H3** (Sec. 1.2): self-supervision through random masking provides adequate training signal for learning abstract reasoning. The learning curves in Section 4.5.2 further support this: metrics saturate during Random masking, then improve when the model fine-tunes on Query masking with representations already acquired.

4.5.2 Learning Dynamics Analysis

To complement the aggregate results in Table 4.2, we examine representative learning curves for a Task tokenizer trained with Combined masking. In this particular run, the Random masking phase stopped at epoch 199 (out of a maximum of 200) based on validation early stopping, and the subsequent Query-only fine-tuning phase terminated after 12 epochs, again via early stopping.



(A) **Correct** and **PropRate** across training epochs.

(B) **PropAcc** on the currently masked panel only.

FIGURE 4.5: Learning curves for the Task tokenizer under Combined masking (Random masking for 199 epochs, then Query-only fine-tuning for 12 epochs). The phase transition creates a visible discontinuity in metrics that aggregate classification and prediction performance.

Figure 4.5a reports **Correct** and **AvgProp** on both training and validation sets. During the Random masking phase (epochs 1–199), these curves reflect a mixture of *classification* (identifying properties of visible context panels) and *prediction* (forecasting properties of the currently masked panel). The final segment captures the Query phase, where evaluation switches to prediction-only. The apparent drop at the phase boundary simply reflects the increased difficulty of prediction compared to classification, without implying any loss of learned capability. The Query-only phase still yields measurable gains; however, the short duration of the second phase (12 epochs) indicates that most representation learning occurs during Random masking, with the Query phase primarily refining the learned representations.

Figure 4.5b reports an analogous plot for the same training run, now in terms of **PropAcc**. This metric represents the average accuracy across individual properties: arrangement, position, type, size, and color. While PropAcc resembles AvgProp, differences arise because panels vary in the number of relevant objects and properties, preventing exact equivalence between the two metrics. Here, PropAcc is

computed exclusively for the currently masked panel, isolating prediction quality from the classification component.

The early training trajectory in Figure 4.5b is less steep than in Figure 4.5a, since the metric does not benefit from the comparatively easier *classification* component. The Random-to-Query transition again yields observable improvement: validation **PropAcc** continues to increase for several epochs in the Query phase even after flattening during Random masking, consistent with query-focused fine-tuning extracting additional accuracy from representations learned in the first phase. The Query-only phase exposes stronger overfitting: training **PropAcc** rises rapidly while validation **PropAcc** peaks and declines around epoch 12, triggering early stopping—in contrast to the more gradual separation during Random masking, where diverse prediction targets act as an implicit regularizer. This progression loosely resembles *curriculum learning*: the model first stabilizes representations through the easier classification objective, then shifts full optimization toward the harder prediction target that matches downstream evaluation. Empirically, Figures 4.5a and 4.5b capture this handoff, with validation improvements during query-focused fine-tuning accompanied by stronger overfitting pressure once Random masking is removed.

4.5.3 Tokenizer Performance Patterns

Given that Combined masking works best across all tokenizers, we focus on this configuration to compare tokenization strategies. The Row tokenizer achieves the highest prediction accuracy (77.58% Correct; Table 4.2), followed by Task (75.63%) and Panel (22.17%). Classification accuracy shows the opposite pattern: Panel excels (98.28%; Table 4.3), while Row and Task achieve 87–88%.

Table 4.3: Property classification accuracy on directly observable context panels.

Tokenizer	Masking	Correct	PropRate	AvgProp	AvgH
Panel	Query	97.54	99.53	99.50	0.05
	Random	98.38	99.49	99.44	0.05
	Combined	98.28	99.53	99.49	0.05
Task	Query	89.69	98.28	98.35	0.31
	Random	88.44	97.90	98.00	0.37
	Combined	88.33	97.87	97.98	0.38
Row	Query	84.03	97.70	97.68	0.37
	Random	87.64	98.17	98.22	0.31
	Combined	87.85	98.20	98.25	0.31

This inverse relationship between classification and prediction performance (Table 4.3) suggests a tradeoff. Panel tokenization optimizes for classifying individual panels but sacrifices the ability to compare across panels. Row and Task tokenization enable cross-panel reasoning but at the cost of some classification accuracy. Since our goal is property prediction—reconstructing the missing panel rather than merely identifying visible ones—architectures that facilitate cross-panel comparison prove more advantageous despite lower classification scores.

The Row tokenizer achieves the best prediction performance while being the most computationally efficient (Table 4.4). It uses only 27 tokens compared to 81 for Panel and 64 for Task, leading to fewer attention operations in the transformer. Despite nearly identical transformer parameter counts (2.64–2.65M), the Row tokenizer requires fewer FLOPs: 29 MFLOPS for transformer operations versus 88 for Panel and 53 for Task. The total computational cost follows the same pattern, with Row requiring only 794 MFLOPS compared to Panel's 2326 and Task's 2002. This efficiency gain stems directly from token count—attention complexity scales quadratically with sequence length, making shorter sequences

cheaper. The Row tokenizer thus demonstrates that *how* tokens are arranged matters more than *how many* tokens are used, achieving superior performance with lower computational cost.

Table 4.4: Computational characteristics of tokenizer architectures.

Tokenizer	#parameters [M]		MFLOPS	
	Transformer	Total	Transformer	Total
Panel	2.65	11.45	87.77	2326.16
Task	2.65	11.19	52.84	2002.38
Row	2.64	10.68	29.02	793.97

4.5.4 Why Tokenization Strategy Matters: Architectural Analysis

The performance differences between tokenizers stem from when and where cross-panel comparisons can occur in the model architecture.

Panel Tokenizer: Isolated Processing. The Panel tokenizer processes each of the nine panels independently through the CNN feature extractor, so no information flows between panels within the extractor. Cross-panel comparisons can only occur later, in the transformer layers, after panels have already been encoded into abstract feature representations.

Classification accuracy reaches 98.28% because the task requires only local feature extraction. For prediction, however, the model must compare properties across panels to detect patterns, and the Panel tokenizer can only perform such comparisons in high-level transformer representations, not in the low-level CNN features where properties like color and size are most explicit. This architectural bottleneck explains why Panel prediction accuracy (22.17%) lags far behind classification (98.28%). As shown in Section 4.5.5, this limitation particularly affects low-level properties like color and size, while the Panel tokenizer performs relatively better on high-level features like Type, which can be more easily compared in abstract representations.

Row Tokenizer: Channel-Aligned Comparison. The Row tokenizer takes a different approach. It stacks the three panels from each row as separate channels of a single image—treating them like the red, green, and blue channels of a color photograph. The CNN processes each row-stack independently, creating tokens where spatially corresponding objects from different panels appear at the same position but in different channels.

This channel-wise alignment makes cross-panel comparison straightforward. The CNN can compare property values across channels at the same spatial location, exactly what convolutional filters do when processing RGB images. The model can detect patterns directly in the low-level CNN features, before any high-level reasoning occurs.

Dual Processing Hypothesis. We hypothesize that this architectural advantage enables *dual processing*, where classification and prediction operate synergistically rather than in isolation. Even during classification, when the model has direct access to panel images, we observe that it uses cross-panel information to verify its predictions. Classification errors in the Row tokenizer often resemble rule misapplication—as if the model inferred a panel's properties from neighboring panels and applied the wrong rule—rather than simple perceptual mistakes. This indicates the model simultaneously extracts properties from individual panels and uses relational patterns across panels, with the two processes reinforcing each other through a verification mechanism.

The Panel tokenizer can engage in dual processing, but only at the transformer level after features have been abstracted. The Row tokenizer benefits from dual processing already in the CNN extractor, where low-level features are directly accessible. This earlier integration of classification and prediction signals

provides stronger learning and more robust representations, explaining the Row tokenizer's superior performance.

Relationship to Hypothesis H1. This dual processing hypothesis might seem to contradict **H1** (Sec. 1.2), which advocates for task decomposition. However, **H1** concerns the separation between *property prediction* (Stage 1) and *answer selection* (Stage 2), not the relationship between classification and prediction within Stage 1. Both classification and prediction of panel properties occur in the same stage, and their cooperation strengthens that stage's performance.

Our position on decomposition is nuanced. We believe demanding reasoning benefits from breaking tasks into stages, but we do not advocate for complete isolation of components. Instead, components should maintain feedback loops where solving one objective helps solve another. This cooperative decomposition represents a middle ground: structured enough to encourage learning of useful subtasks, yet flexible enough to allow synergies to emerge. This philosophy likewise underlies our *pseudosymbolic processing* approach, where we use architectural biases to encourage symbolic-like representations without rigidly enforcing them, allowing the model freedom to discover viable solutions.

4.5.5 Analysis of Error Structure in Property Prediction

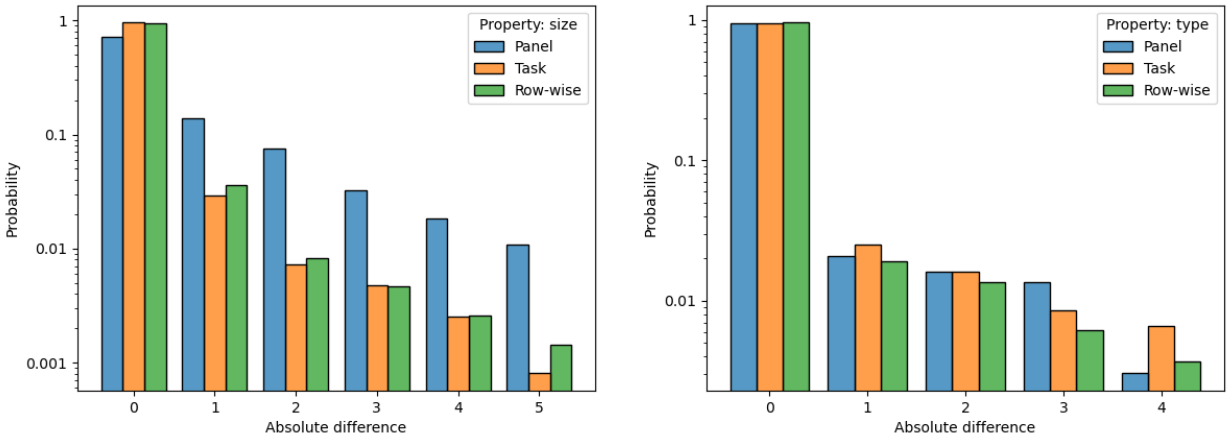
Having established how tokenizer architectures differ in their capacity for cross-panel comparison, we now examine the quality of learned representations by analyzing prediction errors. While RPM is formally an abstract reasoning task, its representation is inherently visual (Raven, 1936), requiring models to interpret low-level visual features and translate them into abstract symbolic properties. These patterns vary across tokenizers in ways that align with the architectural constraints identified above.

Ordinal Structure Discovery in Property Representations. An informative perspective emerges from analyzing the structure of errors during property prediction for objects, specifically regarding color, size, and type attributes. The first two properties possess intrinsically ordered domains, while type values can be ordered by the number of edges: triangle (3) < square (4) < pentagon (5) < hexagon (6) < circle (∞). This ordering enables us to define a metric of **absolute difference** between the predicted position in the property's domain and the actual position. Figure 4.6 presents histograms of this absolute difference error (Mean Absolute Error on the ordinal indices), calculated on the test set for models trained using the Combined masking mode, with absolute difference of 0 occurring at high probability—note the logarithmic scale of the probability axis[1].

Emergent Understanding of Ordinal Scales. The probability of committing errors decreases monotonically with increasing absolute difference values across nearly all error magnitudes. This pattern emerges despite our models treating properties as *categorical distributions* using standard multi-class classification, where we compare the predicted multinomial probability distribution with the corresponding ground-truth one-hot encoded target vector. Under this formulation, all incorrect predictions receive equal penalty: predicting a nearby value versus a distant value incurs the same loss. The **ordinal nature** of these properties (that certain colors, sizes, or types are "closer" to each other than others) was neither explicitly revealed during training nor encoded in the architecture. The models discovered these ordinal relationships purely through self-supervised exposure to visual patterns. The monotonic decrease on a logarithmic scale provides evidence for **H3** (Sec. 1.2): properly structured self-supervision can induce sophisticated conceptual understanding beyond the explicit supervision signal, confirming that models successfully relate low-level visual features to high-level abstract properties—a foundational component of abstract reasoning.

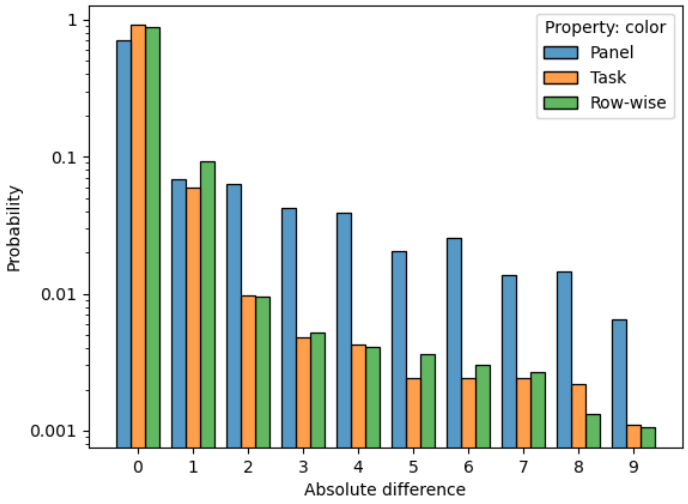
[1] PropRate in Table 4.2 encompasses all properties including arrangement and object presence, while Figure 4.6 focuses exclusively on object properties.

We hypothesize that these models would generalize beyond the training domain. Limited evidence emerges from PGM dataset experiments (Chapter 6), where models recognized squares larger than any in RAVEN. However, rigorous out-of-distribution evaluation faces a practical constraint: RAVEN property values nearly exhaust the feasible range. Colors span the spectrum uniformly, sizes cover the practical detection range (smaller figures become imperceptible; larger ones appear only modestly in PGM), and types progress from triangle to circle (defined as maximum edges). Systematic out-of-distribution analysis would require either expanding property ranges or artificially constraining training exposure to create controlled test conditions.



(A) Size

(B) Type



(C) Color

FIGURE 4.6: Histograms of absolute difference errors committed by the models on object properties. Analogous distributions for untrained (random) models (not shown for clarity) are approximately uniform, confirming that the peaked distributions reflect learned ordinal structure.

Tokenizer-Specific Analysis. The ordinal error patterns described above hold across all three properties and model configurations; however, error distributions for non-zero errors are most uniform for the type property, suggesting that discovering and encoding ordinal relationships for this attribute present the greatest challenge. The Panel tokenizer exhibits a notable pattern (Figure 4.6): while the monotonic decrease trend holds partially, it performs noticeably worse for Color and Size properties, yet this does not extend to the Type property. As discussed in Section 4.5.4, this discrepancy stems from the Panel to-

kenizer's architectural constraint of analyzing individual panels in isolation. The model initially processes images to detect object properties within each panel, enabling subsequent relationship analysis primarily through transformer components. This approach proves most viable for Type property prediction, as type represents a higher level of abstraction requiring dedicated neural processing to generate appropriate representations. Conversely, color and size represent lower-level visual attributes that benefit from direct cross-panel comparison early in the processing pipeline. The Panel tokenizer, having access only to single panels during CNN feature extraction, lacks the capability for preliminary inter-panel comparisons that benefit basic property analysis. The error patterns observed here directly reflect this architectural limitation: the model struggles precisely with the properties that require low-level feature comparison, confirming the architectural analysis from Section 4.5.4.

4.5.6 Cross-Benchmark Validation and Generalization

Tables 4.5 and 4.6 present performance on I-RAVEN and RAVEN-FAIR, two benchmark variants designed to eliminate different types of dataset bias (Sec. 3.2.2 and Sec. 3.2.3). All models were trained on RAVEN and evaluated on I-RAVEN and RAVEN-FAIR without retraining. Since property prediction operates solely on context panels—which are generated identically across all three benchmarks—the datasets differ only in answer set construction, which our property prediction stage does not access. The later experiments on choice-making models, which do use answer sets, show similar transfer results, providing additional validation for the findings in this section.

Table 4.5: Property prediction performance on I-RAVEN, which eliminates answer distribution biases through attribute bisection trees.

Tokenizer	Masking	Correct	PropRate	AvgProp	AvgH
Panel	Query	18.29	72.67	69.86	3.54
	Random	21.76	83.30	81.20	2.15
	Combined	25.20	84.75	82.82	1.99
Task	Query	37.45	86.34	84.96	1.89
	Random	65.38	94.85	94.14	0.81
	Combined	71.18	95.67	95.07	0.71
Row	Query	38.75	85.02	83.80	2.14
	Random	75.99	96.17	95.73	0.63
	Combined	78.95	96.63	96.21	0.56

Table 4.6: Property prediction performance on RAVEN-FAIR, which mitigates attribute-rule correlations through dynamic sampling.

Tokenizer	Masking	Correct	PropRate	AvgProp	AvgH
Panel	Query	16.48	71.38	68.40	3.67
	Random	21.89	83.19	81.10	2.16
	Combined	25.29	84.62	82.72	2.01
Task	Query	35.05	85.55	84.08	1.98
	Random	64.21	94.60	93.87	0.84
	Combined	70.09	95.49	94.89	0.74
Row	Query	36.50	84.46	83.15	2.19
	Random	74.87	95.98	95.55	0.66
	Combined	77.88	96.49	96.07	0.59

The results demonstrate strong generalization. The Row tokenizer with Combined masking achieves 77.58% on RAVEN, 78.95% on I-RAVEN, and 77.88% on RAVEN-FAIR—nearly identical performance

across all three benchmarks. The Task and Panel tokenizers show similar consistency. The ranking of tokenizers (Row > Task > Panel) and masking strategies (Combined > Random > Query) remains identical across benchmarks. Even the magnitudes are comparable: PropRate stays around 95–96% and AvgH around 0.56–0.74 for the best configurations. This consistency is significant because I-RAVEN and RAVEN-FAIR each use a different answer set construction strategy to eliminate answer-set biases (Sec. 3.2.3): models exploiting answer set distributions, positional biases, or attribute-rule correlations would fail when these patterns change across benchmarks—and they do not. This confirms **H2** (Sec. 1.2) on bias mitigation through property prediction, and supports **H1** (Sec. 1.2) by demonstrating that separating property prediction from choice making yields representations that transfer across datasets rather than exploiting RAVEN-specific shortcuts.

4.6 Architecture Validation: Transformer vs. DenseFormer

4.6.1 Architecture Comparison

A central question underlying our approach concerns whether the transformer architecture provides tangible advantages for abstract reasoning tasks. Specifically, does learning to model direct interactions between tokens representing different panels outperform simpler feedforward architectures?

To address this empirically, we implemented **DenseFormer** baselines where the transformer is replaced with dense feedforward layers. Tokens produced by the image tokenizer are concatenated into a single vector and processed through five dense layers of equal size. The final layer output undergoes the same partitioning and post-processing as ACT: split into nine chunks for predicting individual panel properties. Since the Row tokenizer achieved the best performance (Table 4.2), we designed two DenseFormer variants with 336 and 512 units, respectively, matching ACT's parameter count and computational cost (Tables 4.7 and 4.4).

Table 4.7: Computational characteristics of DenseFormer baseline models.

Dense layer size	#parameters [M]		MFLOPS	
	Transformer	Total	Transformer	Total
336	2.67	10.70	5.34	770.23
512	4.33	12.37	8.67	773.57

4.6.2 Results

Each variant was trained in Random masking mode under two configurations: with and without layer normalization (Ba et al., 2016) and dropout. Table 4.8 shows a stark performance gap: DenseFormers fail to approach usable performance, with Correct below 1% across all configurations. For comparison, the Row tokenizer with Combined masking achieves 77.58% (Table 4.2)—nearly two orders of magnitude better than DenseFormer's best result of 0.88%. Even property-level metrics reveal severe deficiencies: PropRate reaches only 48–56% for DenseFormers versus 96% for ACT, while AvgH remains above 5.3 compared to ACT's 0.56.

Table 4.8: Performance comparison of DenseFormer baseline models against ACT variants.

Dense size	Layer normalization	Correct	PropRate	AvgProp	AvgH
336	No	0.24	48.41	43.08	6.03
	Yes	0.45	55.19	51.24	5.43
512	No	0.28	49.76	44.76	5.88
	Yes	0.88	55.92	52.03	5.36

Increasing layer size from 336 to 512 yields similarly marginal gains, suggesting that simply scaling up feedforward capacity would not bridge this performance gap. Interestingly, layer normalization provides consistent improvement despite the models' catastrophic overall performance, raising Correct from 0.24% to 0.45% (336 units) and from 0.28% to 0.88% (512 units). Moreover, validation curves for normalized models showed gradual improvement over time rather than plateauing immediately. Without normalization, models performed worse and stagnated at higher error rates earlier in training. While neither configuration approaches usable performance, the contrast confirms normalization as a universally beneficial technique, improving performance regardless of underlying architectural adequacy.

4.6.3 Why Transformers Matter for Abstract Reasoning

The catastrophic failure of DenseFormers reveals that **cross-talk between tokens**—the defining capability of transformer architectures (Sec. 4.2.3)—proves essential rather than merely beneficial for abstract reasoning in RPMs. Feedforward networks, despite matching parameter count and computational budget, cannot solve the task because they lack mechanisms for relating information across different panels. Pattern recognition in RPMs requires detecting differences between objects that only become apparent when compared across panels; without attention, this cross-panel comparison is structurally unavailable.

The attention mechanism enables this relational reasoning by identifying salient elements and establishing connections across panels. By selectively focusing on informative distinctions, attention allows the model to treat specific elements as premises for inference while ignoring spurious information—for example, detecting a shape in the leftmost panel may trigger attention toward corresponding shapes in other row panels, facilitating systematic pattern recognition.

This cross-talk capability also helps explain why our self-supervised training approach is well suited. Attention facilitates *bidirectional information flow* across panels: visible panels inform masked panel predictions through learned relationships, and since any panel can be masked, the model cannot rely on position-specific features but must learn relational patterns capturing the underlying logical structure. This sharpens the interpretation of **H3** (Sec. 1.2): self-supervised property prediction provides an adequate training signal when combined with architectures capable of cross-panel reasoning—directly instantiating the two-component framework of implicit symbolic processing (Sec. 2.4.3, Sec. 2.4.3).

4.7 Data Augmentation Analysis

Seeking better generalization, we evaluate data augmentation strategies for ACT. We implement three matrix-level transformations applied during training (Fig. 4.7):

- Random permutation of rows.
- Random permutation of columns.
- Transposition of the panel matrix.

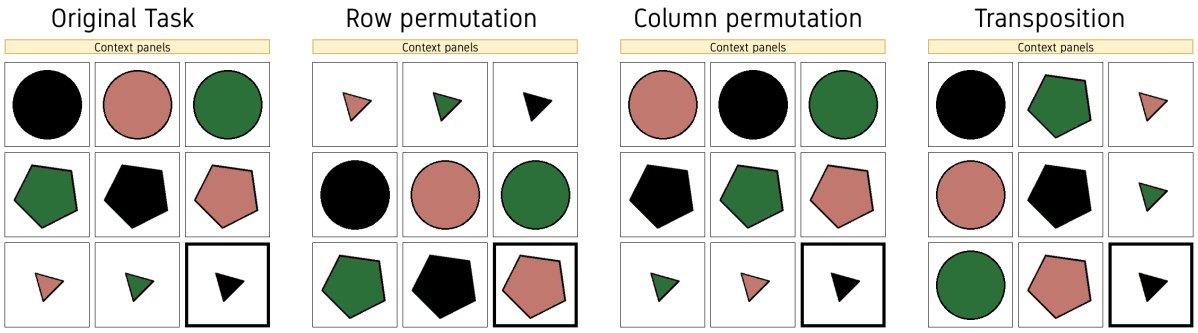


FIGURE 4.7: An RPM task completed with the correct answer panel (left) and its augmented versions obtained by random permutation of rows, random permutation of columns, and transposition of the panel matrix.

When presenting an RPM to the model, each augmentation is applied with independent probability, allowing augmentations to co-occur. We evaluate seven probability combinations across three training modes: augmentation only during the Random masking phase, only during the Query masking phase, or during both phases. These experiments were conducted using the Task tokenizer with Combined masking; the choice of tokenizer for choice-making experiments is discussed in Chapter 5.

Due to the inherent structure of the RAVEN dataset and our property prediction task framework, *row permutation* does not violate any underlying rules, and the task remains fully solvable. This is because RAVEN tasks apply rules exclusively in a row-wise manner (Zhang et al., 2019a) (Sec. 3.1.1), making row reordering a semantically valid transformation that preserves the logical structure while providing beneficial training diversity. Conversely, column permutation and transposition can potentially disrupt the rule structure, rendering the target panel incorrect for the intended solution. For example, column permutation can break *progression rules* by altering the sequential order required for pattern completion. Despite this potential for rule violation, we conducted experiments with these transformations, treating them as regularization techniques that could enhance model resilience and reduce overfitting tendencies (Zhang et al., 2018).

In preliminary experiments, we evaluated random panel replacement as an additional regularization strategy. However, this approach consistently decreased performance across all experimental configurations, leading us to exclude this method from our final augmentation pipeline.

4.7.1 Experimental Results

Table 4.9 and Fig. 4.8 reveal several patterns. First, configurations E, F, and B—which use only row permutations—systematically fare best, achieving 76.30–78.60% accuracy. Configuration E (70% row permutation during Random phase) improves 3 percentage points over the baseline (75.63%). In contrast, configurations incorporating column permutation or transposition (A, C, D) perform worse, achieving only 65.75–73.88% in their best variants. Contrary to our expectation that rule-breaking transformations might still act as useful regularizers, this performance degradation suggests that violating the row-wise rule structure of RAVEN tasks introduces noise that impairs rather than enhances learning.

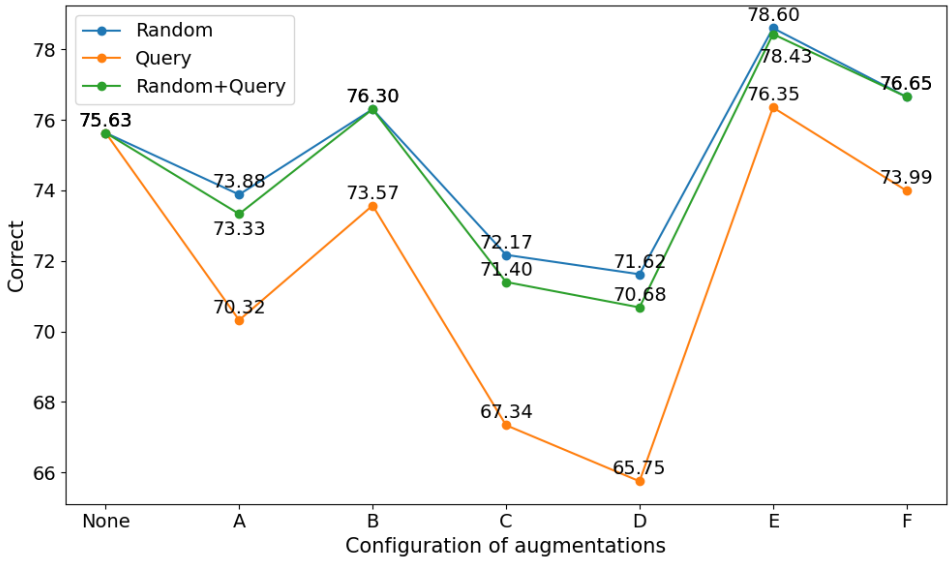


FIGURE 4.8: Impact of augmentations on ACT's accuracy (metric: Correct).

TABLE 4.9: Impact of augmentations on property prediction accuracy. Probabilities represent (row, column, transpose) permutation rates. Columns show which training phase(s) used augmentation.

Probabilities			Tag	Correct (%)		
row	col	trans		Random	Query	Random+Query
0.0	0.0	0.00	None	75.63		
0.2	0.1	0.05	A	73.88	70.32	73.33
0.4	0.0	0.00	B	76.30	73.57	76.30
0.4	0.2	0.10	C	72.17	67.34	71.40
0.6	0.3	0.15	D	71.62	65.75	70.68
0.7	0.0	0.00	E	**78.60**	76.35	78.43
1.0	0.0	0.00	F	76.65	73.99	76.65

Second, the relationship between row permutation probability and performance is non-monotonic. Performance improves from 40% (B: 76.30%) to 70% (E: 78.60%), but decreases when always applying row permutation (F: 76.65%). At probability 1.0, every sample receives random row permutation uniformly distributed across all possible orderings, whereas at 0.7, the model sees 70% randomly permuted and 30% in original order. We speculate that the superiority of 0.7 over 1.0 may indicate subtle bias or non-uniform distribution in the original RAVEN data, where certain properties, rules, or patterns are not evenly distributed across row positions. The model appears to benefit slightly from exposure to samples in their canonical ordering. However, any such bias is likely small, given that the probability is already quite close to uniform distribution.

Third, applying augmentations during the Query masking phase is generally detrimental or neutral at best, whether used alone or combined with Random phase augmentation. However, drawing firm conclusions about the underlying reasons is challenging given the mixed patterns in the data. The detrimental effect is most pronounced in configurations with column permutation or transposition (A, C, D), which already show negative impacts on results generally. For row-only permutation configurations (B, E, F), Random and Random+Query yield nearly identical results—two configurations are exactly equal (B: 76.30%, F: 76.65%) and one differs marginally (E: 78.60% vs. 78.43%). When comparing Query-only augmentation against the baseline (no augmentation: 75.63%), row-permutation configurations show mixed results: two perform worse (B: 73.57%, F: 73.99%) and one performs better (E: 76.35%). Given

that the Query phase is only the fine-tuning part of training, and the row-only augmentation yields mixed results for Query augmentation, it is difficult to speculate on the underlying mechanisms. In summary, the results demonstrate that augmentation provides maximum benefit during exploratory learning (Random phase) but offers no clear advantage and a potential disadvantage during specialized fine-tuning (Query phase).

The expected regularization benefit of rule-breaking augmentations did not materialize: all configurations with column permutation or transposition degraded performance. We attribute this to *random masking* (Sec. 4.3.1) already serving as a more general augmentation mechanism (varying prediction contexts across the entire matrix structure), which explains why even valid row permutation yields only modest additional gains: random masking already captures much of the diversity that row permutation provides. These results further provide evidence for **H1** (Sec. 1.2): this training flexibility arises from decomposing the task into property prediction and answer selection—no additional mechanism is needed to enable it. As established in Sec. 4.3.1, decomposition means masking is not restricted to the query panel; we can therefore mask any panel and permute rows without invalidating targets, whereas a monolithic end-to-end framework could only mask the query panel and could not permute rows without invalidating the answer set.

4.8 Analysis of ACT's Learned Representations

Beyond quantitative performance metrics, we investigate how ACT internally represents abstract structure. We focus on the *chunks* of output tokens the transformer produces in response to context panels, which the property predictor maps to property vectors (Figure 4.1). ACT's transformer partitions output sequences into 9 chunks corresponding to the 9 panels. The Row tokenizer generates 27 tokens (3 per chunk) and the Task tokenizer produces 64 tokens (7 per chunk), as defined in Sec. 4.2.3. With 128-dimensional token vectors, chunks span 384-dimensional (Row) or 896-dimensional (Task) latent spaces.

4.8.1 t-SNE Analysis of Learned Representations

We use t-SNE visualization (Maaten and Hinton, 2008) to examine how tokenizer architectures shape learned representations. After querying ACT on the test set, we collect the resulting chunks (including those for the query panel), apply t-SNE dimensionality reduction to two dimensions, and label points by input panel properties.

Property Disentanglement Across Tokenizers. Figure 4.9 presents t-SNE embeddings colored by three object properties (Color, Size, Type) across all three tokenizer architectures. To simplify the analysis and avoid complications from multiple figures per panel, we restricted this visualization to samples with *center-single* arrangement, where each panel contains exactly one object. This choice additionally reduces computational complexity for t-SNE projection.

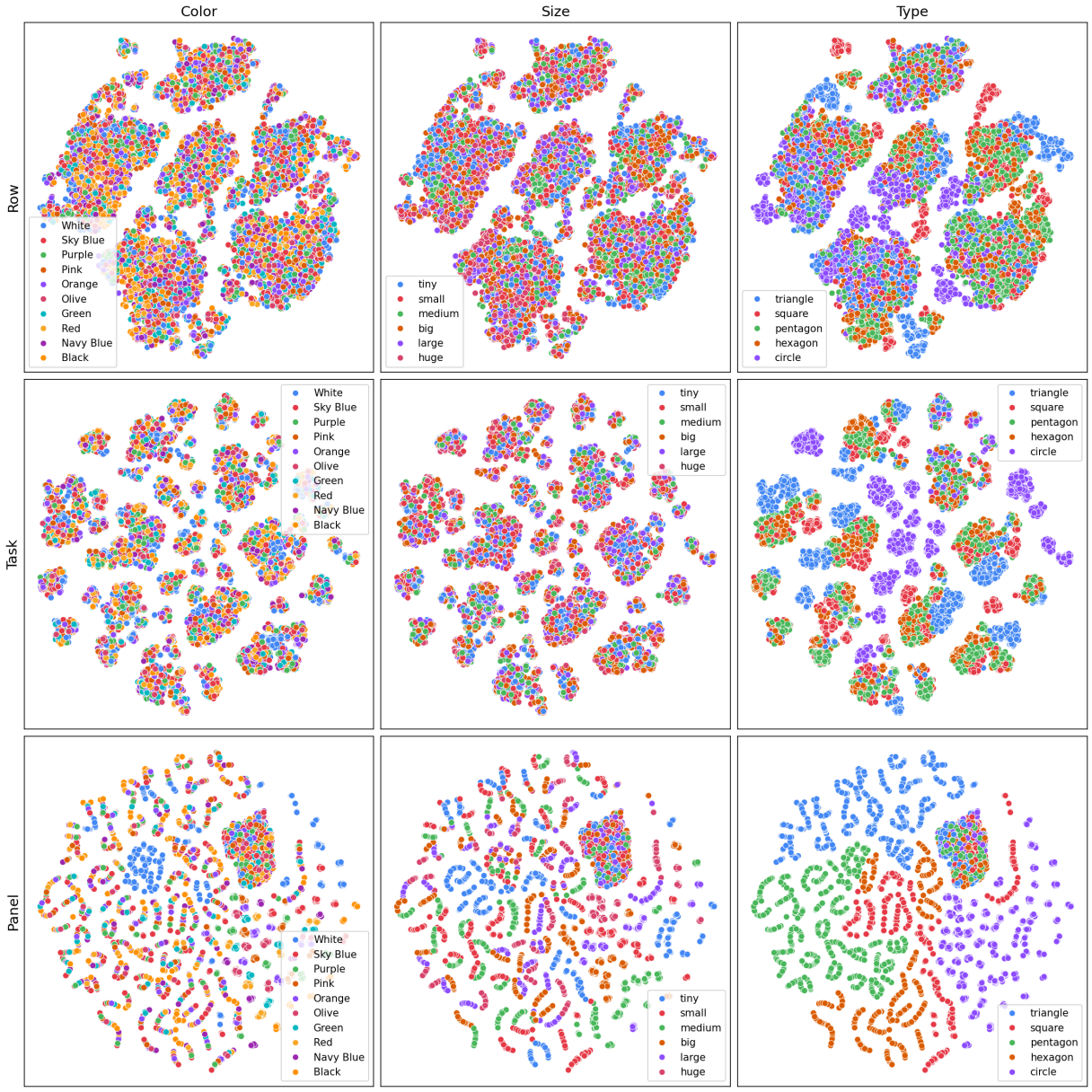


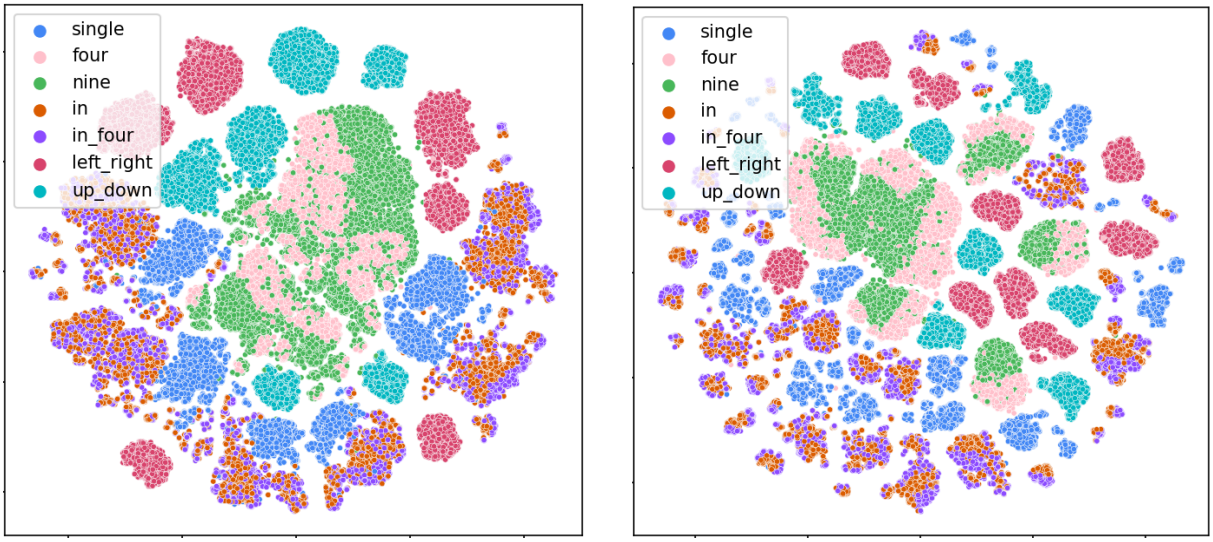
FIGURE 4.9: t-SNE embeddings of transformer output chunks colored by object properties (Color, Size, Type) across three tokenizer architectures (Panel, Row, Task). Each subplot shows chunks from the test set, demonstrating varying degrees of property disentanglement. The Panel tokenizer exhibits clear separation for all properties, particularly Type. Row and Task tokenizers show disentanglement primarily for Type, with Color and Size forming less structured clusters due to early-stage inter-panel comparison emphasis.

The Panel tokenizer processes individual panels at feature extraction and cannot perform preliminary inter-panel comparisons. This separation produces disentangled representations optimized for classification tasks. Visualizations confirm clear disentanglement for Type, strong separation for Size, and moderate separation for Color. This hierarchy reflects the varying abstraction levels required for different property types.

Task and Row tokenizers exhibit different behavior. Only Type properties form some level of disentangled representations, while Color and Size show minimal clustering structure or lack it entirely. However, the Task tokenizer shows visibly better disentanglement than the Row—the clusters are smaller and more compact, particularly for Type—suggesting a hierarchy of representation quality even among prediction-oriented architectures.

These differences follow directly from the architectural constraints analyzed in Sec. 4.5.4: Panel tokenizer's isolated panel processing forces property identification before any cross-panel comparison,

yielding clean per-property clusters; Row and Task tokenizers encode inter-panel relationships from the first CNN layers, which sacrifices property disentanglement but benefits prediction performance.



(A) Row tokenizer (B) Task tokenizer

FIGURE 4.10: t-SNE embeddings colored by arrangement for Row (A) and Task (B) tokenizers. Chunks cluster according to arrangement values, with Row producing larger, better-defined clusters due to processing three panels per row. Both tokenizers notably group inside-out arrangements (*in*, *in-four*) together, capturing abstract spatial relationships.

Arrangement Clustering in Row and Task Tokenizers. Given the weaker object-property disentanglement in Row and Task tokenizers, we investigated whether higher-level properties like *arrangement* might reveal additional structure. Since the Panel tokenizer already achieves strong disentanglement for individual object properties, we focused on Row and Task tokenizers. For this analysis, we generated t-SNE embeddings with points labeled by arrangement property, including all samples from the test set covering all arrangement types. Figure 4.10 shows the results.

The embeddings reveal that chunks cluster consistently with arrangement values. The Row tokenizer produces larger, better-defined clusters, while the Task tokenizer generates smaller, more fragmented ones: the Row tokenizer processes only 3 panels at once (a single row), allowing it to more easily disentangle their spatial arrangements, while the Task tokenizer processes all 9 panels simultaneously, increasing representational complexity and fragmenting the clusters. This inverts the property disentanglement pattern—Row achieves the weakest separation in Figure 4.9 but the strongest arrangement disentanglement in Figure 4.10—and reflects the same architectural tradeoff from Sec. 4.5.4: channel-wise stacking biases Row toward encoding spatial configurations early, which becomes an advantage for relational properties like arrangement.

Importantly, this inversion is consistent across both sample sets: the property visualization used only *center-single* samples while the arrangement visualization includes all arrangement types, yet the relative cluster characteristics (Row: larger and more diffuse; Task: smaller and more compact) remain stable. This suggests architectural differences create representational properties that hold independently of the specific data subset analyzed.

Within the arrangement embeddings, several patterns emerge. Configurations with few objects (*single*, $k = 1$; *left_ right*, *up_ down*, $k = 2$) form clean, well-separated clusters. Arrangements with more objects produce mixed clusters. *Distribute-nine* conflates with *distribute-four* because both distribute multiple objects over fixed grids, though these combined clusters show internal partitioning. One pattern stands out: arrangements involving inside-out relations (*in* and *in-four*) cluster together. The transformer captured this abstract spatial concept while abstracting from specific object counts—a form of relational

reasoning showing the model's ability to learn semantic similarities beyond surface-level features.

Beyond the main analyses, additional visualization experiments revealed two minor patterns worth noting. Individual tokens within chunks showed marginal Type-discrimination differences, suggesting slight functional specialization, though too small to warrant detailed analysis. Rule-type visualizations showed clear discrimination between *constant* and *non-constant* rules across all tokenizers, but minimal separation among non-constant rule types. This may appear paradoxical given the evidence for cross-panel relationship encoding: if the model captures how properties change across panels, one might expect it to represent the higher-level rules governing those changes as well. However, rules such as *progression* or *arithmetic* are abstractions over low-level changes—the model likely encodes that a figure's color shifts from one value to another, not that this shift follows a named progression pattern. The constant versus non-constant discrimination is consistent with this: it reflects whether any change occurs at all, which is a lower-level signal than identifying which rule produces it. Fully characterizing this granular relationship encoding represents a direction for future work.

4.8.2 SHAP Attribution Analysis

t-SNE visualizations show how chunk representations cluster, but they do not reveal whether the transformer discovered *structural encoding*—whether specific tokens in output sequences convey particular aspects of perceived panels.

We address this by analyzing the Task tokenizer chunks using SHapley Additive exPlanations (SHAP) (Lundberg and Lee, 2017) from the SHAP library.[2] SHAP estimates the importance (attribution) of individual inputs for given predictions through *local explanations*. Here, inputs are the 896 dimensions of the Task tokenizer chunks, and predictions are properties produced by ACT. We analyze the *arrangement* property, which exhibited visible clustering in t-SNE embeddings.

Figure 4.11 presents chunk dimension importance using heatmaps with 7 rows (tokens) and 128 columns (token dimensionality). The top 7 heatmaps show importance for particular arrangement predictions (*single* through *up_down*), followed by element-wise average and standard deviation.

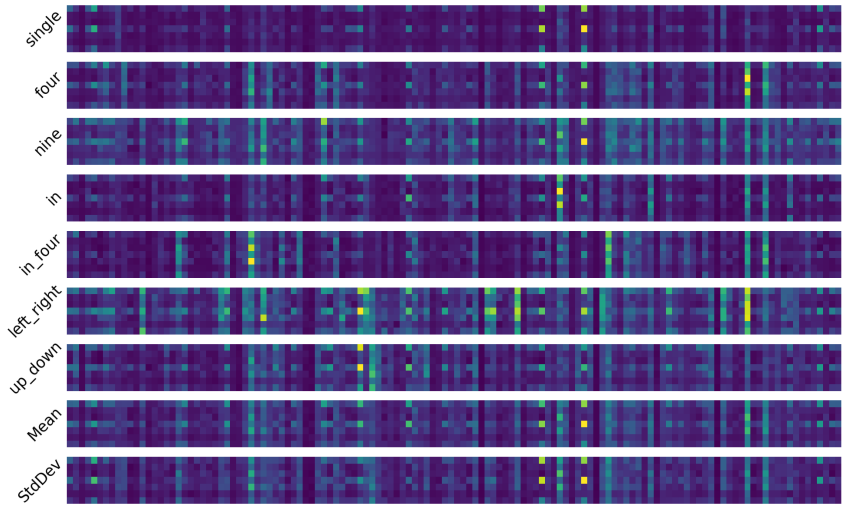


FIGURE 4.11: SHAP attribution values for the 7 × 128 dimensions of Task tokenizer chunks, analyzed by arrangement prediction. Sparse activation patterns indicate localized, semi-symbolic encoding where specific token dimensions encode specific properties.

The heatmaps reveal sparse activation patterns. For any given arrangement prediction, only a small fraction of the 896 dimensions show high importance—most dimensions contribute little or nothing.

[2] `https://shap.readthedocs.io/`

This sparsity holds both at the chunk level (across all 896 dimensions) and within individual tokens (across their 128 dimensions). Some dimensions matter broadly across multiple arrangement types, appearing as bright spots that repeat across heatmaps and show high mean relevance. Others activate selectively for specific arrangements. The sparsity is not uniform across tokens: different tokens within each chunk rely on different dimensional subsets, with some dimensions remaining active across all 7 tokens (visible as bright vertical stripes in Figure 4.11) while others activate only in specific tokens. This variation suggests the model distributes arrangement encoding across the token sequence, with each token potentially capturing different spatial or structural aspects. The close resemblance between mean importance and standard deviation patterns indicates that high-importance dimensions tend to activate selectively—those dimensions that matter most for arrangement encoding shift their roles depending on which specific arrangement is being predicted.

ACT could have learned a dense representation where many chunk dimensions contribute to arrangement (and potentially other properties simultaneously). Instead, the transformer expresses arrangement with relatively few selected dimensions in selected tokens. This sparse encoding differs from the distributed representations typical of deep learning and suggests a form of **localized, semi-symbolic encoding**. The model appears to disentangle certain input characteristics and encode them in specific token dimensions rather than distributing information diffusely across the entire representation. Verifying this hypothesis thoroughly would require extensive additional investigation—a promising direction for future work, particularly with the enhanced architecture proposed in Chapter 6, which we expect will exhibit even clearer semi-symbolic structure. If confirmed, such representations would occupy an interesting middle ground between purely symbolic approaches and conventional deep learning—similar in spirit to the Vision Transformer architecture (Dosovitskiy et al., 2020). This interpretability advantage directly reflects **H1** (Sec. 1.2): separating property prediction from choice making creates the conditions under which the model can develop structured, inspectable internal representations rather than opaque end-to-end mappings.

4.9 Summary and Conclusion

4.9.1 Validation of Hypotheses

The experimental results across masking strategies, tokenizer architectures, benchmark variants, and architectural comparisons collectively validate all three hypotheses motivating this chapter (Sec. 1.2).

H1 — Staged Decomposition. Tokenizer architecture determines when cross-panel comparisons can occur: the Row tokenizer's channel-wise stacking enables pattern detection in low-level CNN features, achieving the highest prediction accuracy with the lowest computational cost (Table 4.4), while the Panel tokenizer's isolated processing yields the strongest classification but weakest prediction—an inversion that persists across all masking strategies (Table 4.2). This reflects the *dual processing hypothesis* (Sec. 4.5.4): within Stage 1, classification and prediction operate synergistically rather than in isolation, with the Row tokenizer benefiting from this cooperation already at the CNN level. Crucially, **H1** concerns separating *property prediction* (Stage 1) from *answer selection* (Stage 2), not isolating components within a stage: decomposition between stages, synergy within each stage. This decomposition makes flexible training methodologies possible: random masking can be applied to any panel, and row permutations do not invalidate targets (Sec. 4.7), whereas a monolithic framework would permit neither. The t-SNE and SHAP analyses (Sec. 4.8.1) confirm this picture: staged decomposition creates conditions for interpretable representations, though disentanglement emerges selectively: the Panel tokenizer achieves clear separation for Color, Size, and Type, while Row and Task tokenizers show arrangement clustering but minimal basic-property disentanglement, reflecting their architectural bias toward early relationship encoding.

H2 — Bias Mitigation. Consistent performance across RAVEN, I-RAVEN, and RAVEN-FAIR (Sec. 4.5.6; Table 4.5, Table 4.6), confirms that property prediction successfully mitigates dataset biases introduced by answer set construction (Sec. 3.2.2). I-RAVEN and RAVEN-FAIR each use a different construction strategy to eliminate answer-set biases (Sec. 3.2.3); models exploiting answer set distributions, positional biases, or attribute-rule correlations would fail when these patterns change across benchmarks—and they do not.

H3 — Self-Supervised Training. Random and Combined masking dramatically outperform Query-only masking across all tokenizers (Task tokenizer: 18.91% → 72.63% Correct; Table 4.2): self-supervision through random masking provides adequate training signal for learning abstract reasoning. Learning curves (Sec. 4.5.2) reveal a curriculum-like dynamic: Random masking stabilizes representations through diverse prediction targets acting as an implicit regularizer, after which Query-only fine-tuning extracts additional accuracy while exposing stronger overfitting pressure. The emergence of ordinal structure in prediction errors (Sec. 4.5.5)—where models discover proximity relationships between property values never made explicit during training—further confirms that self-supervision induces conceptual understanding beyond the explicit supervision signal. The DenseFormer results (Sec. 4.6) sharpen this interpretation: self-supervision is adequate only when combined with architectures capable of cross-panel reasoning, confirming that the training regime and the architectural substrate are jointly necessary.

4.9.2 Architectural Reflections

The experimental results shed light on two design choices whose implications were not fully predictable in advance: the adequacy of one-dimensional positional encoding for an inherently two-dimensional task, and the role of the learnable mask in enabling flexible self-supervised training.

Spatial Structure from Sequence Encoding

Even though RPM problems have an inherent 2D structure, our models rely on sequence-to-sequence transformers (Vaswani et al., 2017) with standard one-dimensional positional encoding. The experimental results demonstrate that capable RPM solvers can be obtained without explicitly presenting the spatial structure of the problem to the model—a result that is, at first sight, surprising. We believe the model autonomously learns the *spatial structure* of the task from positional encoding alone. A notable precedent is (Dosovitskiy et al., 2020), where sequential transformers operating on linear sequences of 16×16 image patches excelled at image classification, requiring fewer computational resources than conventional architectures. There is a growing body of evidence that sequences alone provide sufficient perceptual structure for a wide range of tasks.

More specifically, the absolute positional encoding allows the model to identify the origin of each token within the input image. In our preliminary experiments, when we analyzed the learned positional embeddings, the model appeared to acquire the 2D structure of the task through 1D positional encoding.

Alternative methods exist for incorporating spatial information more explicitly into the transformer—for instance, multidimensional attention mechanisms that assess similarity separately along rows and columns. While this presents an intriguing avenue for research, caution is warranted: explicitly encoding spatial structure risks reducing model flexibility and generality, and preliminary experiments with basic 2D structural encoding yielded inferior results compared to the one-dimensional baseline. Providing the model with flexibility to autonomously learn spatial structure is a well-established strategy in deep learning, and analysis of attention patterns confirms that the simple positional embedding approach is sufficient for the model to internalize the puzzle's grid structure.

Mask Value Implementation and Its Implications

In preliminary experiments, we evaluated several approaches for filling the masked panel before tokenization: zeros, noise, merging multiple panels as a composite mask, and curriculum-style schemes transitioning from partial to full occlusion over training steps. The best-performing approach was the **learnable mask**: a single image initialized with random pixel values and optimized jointly with the model during training. Its appearance after training is visualized in Figs. 4.2, 4.4, and 4.3.

The success of the learnable mask carries a broader implication for **H3** (Sec. 1.2). The mask is initialized randomly and optimized end-to-end; despite this, no discernible interpretable pattern emerges in the learned values. This suggests that the mask serves not as an informative signal but as an optimization degree of freedom: it allows the model to settle on input representations that best separate masked from non-masked positions without any hand-crafted design. This exemplifies a general principle underlying **H3** (Sec. 1.2): self-supervised training works best when paired with flexible, adaptable components. Rather than imposing fixed masking values, the learnable approach enables the optimizer to find mask patterns that facilitate reliable feature extraction and pattern recognition, mirroring the rationale for learnable positional encodings.

4.9.3 Conclusion

This chapter demonstrated that decomposing the RPM task into property prediction and choice making yields a model that learns true abstract reasoning capabilities rather than dataset-specific shortcuts, with consistent performance across RAVEN, I-RAVEN, and RAVEN-FAIR providing strong empirical support for **H2** (Sec. 1.2) and **H3** (Sec. 1.2). Chapter 5 presents choice-making modules that build on these validated ACT representations to solve the traditional RPM selection task, completing the staged framework and providing empirical validation of **H1** (Sec. 1.2) that staged decomposition produces better solvers.

Chapter 5

Choice Makers

Having established in Chapter 4 that the **Abstract Compositional Transformer (ACT)** successfully predicts panel properties through self-supervised learning, this chapter completes the staged self-supervision framework by addressing the second stage: answer selection. While property prediction provides robust, bias-resistant representations of abstract reasoning patterns, practical application to RPM benchmarks requires transforming these predictions into choices among the eight available answer panels.

This chapter presents three choice-making modules of increasing sophistication: the **Direct Choice Maker (DCM)**, which applies fixed similarity metrics to compare the property vectors; the **Learnable Choice Maker (LCM)**, which employs trained neural networks to learn adaptive comparison functions; and the **Contrastive Choice Maker (CCM)**, which uses contrastive learning to refine the representation space before applying similarity comparisons.

All three modules share a common inference procedure, illustrated in Figure 5.1. Given a trained model P and a task T, P is first queried on T with the query panel masked out. The 9th property vector p produced in response, corresponding to the query panel, is the model's **prediction** of the answer to the task. Next, for each of the 8 answer panels, $i = 1, \ldots, 8$, P is queried on T with the query panel replaced with the ith answer panel; in each of those queries, the 9th property vector is stored as p_i. This will be referred to as **classification** of a panel (in terms of its properties). Each classification p_i is then paired with the prediction p, forming eight pairs (p, p_i), $i = 1, \ldots, 8$. The three choice makers differ in how they use these pairs to score answer panels and select the solution.

5.1 Direct Choice Maker (DCM)

The arguably simplest approach is the **Direct Choice Maker (DCM)**, which makes decisions by computing the similarity between the *prediction* for the masked query panel and the *classification* of individual answer panels in the same context.

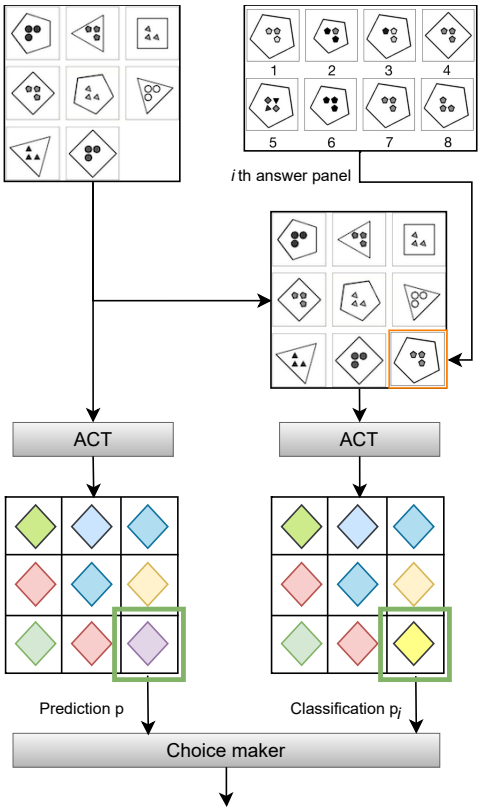


FIGURE 5.1: Overview of the choice-making framework. To solve an RPM task, ACT is queried on context panels alone (prediction p) and on context panels complemented with the ith answer panel (classification p_i, $i \in [1, 8]$). The resulting property vectors are fed into one of the choice makers considered in this chapter: **DCM**, **LCM**, or **CCM**.

5.1.1 Method

DCM selects the answer panel whose classification best matches the prediction: a distance function d is applied to each pair (p, p_i) and the panel with the minimal $d(p, p_i)$ is returned as the solution (Figure 5.1). The distance function takes into account only the relevant properties, where relevance is determined by p_i (the rationale for this choice is discussed in Sec. 4.3.2).

Distance Function and Relevance Source

The performance of DCM depends on two key design decisions: (1) which distance metric d to use for comparing property vectors, and (2) how to determine the relevance of individual properties during comparison.

For the distance function d, we use **cross-entropy**, motivated by the alignment with ACT's training objective: since the model was optimized using cross-entropy loss, the same measure provides a natural and principled basis for comparing its outputs at inference time.

As described in Sec. 4.3.2, not all properties are relevant for every panel due to arrangement differences and uniformity constraints. During comparison, we mask irrelevant properties using relevance information. Since both p and p_i may indicate different relevant properties, we use a single relevance mask derived from the **classification** p_i, ensuring that each property in p aligns directly with its counterpart in p_i and that the number of compared properties remains equal across all answer panels.[1] We choose the classification p_i as the relevance source for two reasons:

[1] Using separate masks for p and p_i would produce misaligned property correspondences and unequal comparison lengths, undermining both computational simplicity and the consistency of the distance computation.

- **Higher accuracy**: Classification operates on a visible panel and is therefore more accurate than prediction, making p_i less prone to errors in determining property relevance (cf. Table 4.2).
- **Conceptual alignment**: p_i serves as the target representation to which the predicted p is compared. Using target-derived relevance focuses the distance computation on properties pertinent to that specific answer panel.

Both choices are corroborated empirically in Sec. 5.1.4.

Evaluation Metrics

We define three performance metrics for evaluating DCM, each measuring the percentage of tasks for which the correct answer panel is selected. The metrics differ in property vector representation and distance function:

- **AccUnique**: Uses categorical property vectors (argmax of probability distributions) with Hamming distance as d. A tie—when $d(p, p_i)$ is minimized by two or more answer panels—counts as a failure.
- **AccTop**: Operates like AccUnique, except that a tie counts as success if the correct answer is among the tying panels.
- **AccProb**: Uses probability distributions produced by the model and computes d using binary cross-entropy for binary properties (e.g., slot occupancy) and categorical cross-entropy for multi-valued properties (e.g., object size, color, type), summed over all relevant properties. This metric leverages the raw distributional output of ACT rather than reducing it to discrete categories.

AccProb serves as our principal metric: it fully utilizes the probabilistic nature of ACT's outputs, aligns with the model's training objective, and benefits from the low likelihood of ties between answer panels, making it sensitive to nuanced, continuous-valued model responses. AccTop and AccUnique are reported as supplementary metrics.

5.1.2 Experimental Results

We evaluated all nine ACT configurations from Sec. 4.4 (three tokenizers × three masking strategies) on the RAVEN test set (Sec. 3.1.1), first establishing *optimistic performance estimates* using ground truth answer properties before reporting actual performance when DCM must classify answer panels itself.

Optimistic Estimates (Results for Ground-Truth Properties)

Optimistic estimates are computed by substituting ground truth property vectors from the RAVEN dataset for the model-classified vectors p_i in step 2 of DCM. Ground truth vectors also determine property relevance, isolating the impact of prediction accuracy on choice-making performance.

For AccProb, this implies comparing the continuously-valued probabilities produced by the model with one-hot vectors representing the categorical values of true properties. Table 5.1 presents the resulting estimates. As expected, AccUnique is the most demanding metric, as it requires the predicted property vector to be uniquely closest to the property vector for exactly one of the answer panels. In contrast, AccTop treats ties as successes and thus reports significantly better scores, though this metric does not reflect the model's capability of pointing to a unique solution among the answer panels.

The relations between models in Table 5.1 correlate with the quality of property prediction (Table 4.2). The Row and Task tokenizers perform comparably in their strongest configurations (Row+Combined: 95.90% and Task+Combined: 95.39%), while the Panel tokenizer lags considerably behind, with Panel

Table 5.1: Optimistic estimates, with DCM relying on *target* property vectors.

Tokenizer	Masking	AccProb	AccTop	AccUnique
Panel	Query	19.00	39.48	6.16
	Random	55.66	70.87	37.47
	Combined	57.57	72.34	39.27
Task	Query	65.09	74.18	38.18
	Random	94.84	96.89	90.70
	Combined	95.39	97.23	92.69
Row	Query	66.45	72.04	36.10
	Random	95.49	96.99	92.62
	Combined	95.90	97.48	94.04

Combined reaching only 57.57%. Across all configurations, Combined masking slightly outperforms Random, and both substantially exceed Query-only masking. These two strongest configurations confirm that accurate property prediction is the dominant driver of choice-making performance when answer panel properties are known precisely.

Results for Predicted Properties

Table 5.2 presents the actual performance of DCM when using *predicted* property vectors p_i obtained by classifying answer panels. While the optimistic estimates (Table 5.1) showed a close correspondence between prediction quality and choice performance, these results reveal marked deviations for most configurations.

Table 5.2: Accuracy on choice tasks, based on *predicted* property vectors. Due to computational resource constraints, multiple runs were performed only for top configurations. $\text{AccProb}_{(n=3)}$ reports mean ± standard deviation over three runs. Preliminary experiments with six runs showed standard deviations of 0.23–0.26 percentage points, corresponding to 95%-confidence intervals of ±0.19–0.21 points, indicating that random initialization has minimal impact on conclusions given the substantial performance differences between models.

Tokenizer	Masking	AccProb	$\text{AccProb}_{(n=3)}$	AccTop	AccUnique
Panel	Query	17.79	—	39.13	6.33
	Random	41.39	—	59.75	25.65
	Combined	46.85	—	63.82	30.74
Task	Query	5.44	—	42.72	0.96
	Random	96.33	95.81±1.47	96.22	83.00
	Combined	96.97	96.45±1.13	96.57	86.30
Row	Query	30.97	—	63.27	5.32
	Random	79.23	80.58±5.67	94.68	25.47
	Combined	82.84	84.66±6.28	95.43	33.53

When the DCM relies on answer panels classified by the ACT, Row+Combined exhibits 82.84% choice accuracy, falling 13.06 percentage points below its 95.90% optimistic estimate, despite maintaining strong 77.58% prediction accuracy (Table 4.2). Panel+Combined shows smaller degradation—10.72 points (46.85% vs. 57.57%)—suggesting that performance gaps between optimistic and actual results differ distinctly across tokenizers. In contrast, Task+Combined reaches 96.97%, *exceeding* its 95.39% optimistic estimate by 1.58 percentage points, indicating that learned representations occasionally provide more precise similarity assessments than direct comparison with ground truth properties.

The optimistic estimates relied on ground truth property vectors rather than model-classified vectors p_i, thereby isolating prediction quality from classification effects. The actual results introduce classification as the sole additional variable, so any performance gap between the two conditions directly reflects

the cost of imperfect answer panel classification. The substantial gaps for Row and Panel tokenizers suggest that their classification of answer panels introduces errors that undermine the similarity-based decision framework, while the Task tokenizer's near-identical performance across both conditions suggests comparatively more accurate classification.

The performance patterns across masking strategies largely follow those observed in optimistic estimates and ACT property prediction. Combined masking consistently outperforms Random masking across all tokenizers (improvements of 3.61 to 5.46 percentage points), confirming that query-specific fine-tuning enhances choice-making as expected. The Query masking strategy similarly produces inferior results, with the Task+Query configuration exhibiting particularly severe degradation (5.44% accuracy) that is analyzed separately in Sec. 5.1.3.

Performance on I-RAVEN Benchmark

Table 5.3 presents DCM performance on **I-RAVEN** (Sec. 3.2.3).

Table 5.3: Accuracy on the test set of I-RAVEN benchmark.

Tokenizer	Masking	AccProb	AccTop	AccUnique
Panel	Query	45.44	64.89	27.81
	Random	53.59	70.22	34.61
	Combined	60.16	74.08	41.24
Task	Query	12.34	51.68	2.50
	Random	94.90	95.42	86.43
	Combined	95.39	95.61	88.95
Row	Query	51.35	76.74	14.30
	Random	86.35	95.10	42.41
	Combined	88.52	95.55	52.09

Apart from Task+Random and Task+Combined, which achieve the best performance on both benchmarks with only slight deterioration on I-RAVEN (96.33%→94.90% and 96.97%→95.39%, respectively), all remaining configurations perform better on I-RAVEN than on RAVEN. Panel+Combined improves from 46.85% to 60.16% (+13.31 percentage points), Row+Combined from 82.84% to 88.52% (+5.68 points), and Row+Random from 79.23% to 86.35% (+7.12 points). The pattern extends even to severely impaired configurations: Task+Query improves from 5.44% to 12.34% (+6.90 points), while Row+Query improves from 30.97% to 51.35% (+20.38 points).

RAVEN and I-RAVEN employ the same context generation procedure (Sec. 3.2.3), so ACT's predictions p and its classifications of the correct answer panels are identical across both benchmarks. Any difference in results between Tables 5.2 and 5.3 must therefore arise from how ACT classifies the seven incorrect answer panels, which are generated differently in each benchmark. RAVEN's biased generation produces incorrect panels that typically differ from the correct answer by a single property—a design intended to challenge human solvers—creating highly similar property vectors; when classification is imperfect, such panels can receive property vectors resembling the prediction p, leading DCM to select them incorrectly. I-RAVEN's incorrect answers diverge from the correct answer more substantially. This reduces the likelihood that classification errors produce property vectors close enough to the prediction p to cause false matches.

The I-RAVEN results offer further evidence for the interpretation suggested in the preceding analysis (Sec. 5.1.2) that answer panel classification is a key factor differentiating tokenizer performance. The gap between Task+Combined and the Row and Panel configurations narrows on I-RAVEN compared to RAVEN, which is consistent with I-RAVEN's more diverse answer sets making answer panel classification

less demanding across all models. If Task+Combined's advantage on RAVEN stems primarily from more accurate classification, that advantage should decrease precisely where classification errors are less prevalent—which is what the I-RAVEN results suggest. Sec. 5.1.3 provides quantitative analysis of these classification patterns and their systematic impact on DCM's choice-making performance.

5.1.3 Classification Performance Analysis

The performance gaps between optimistic estimates and the measured DCM results reported in Sec. 5.1.2 motivate closer examination of how ACT classifies answer panels when they occupy the query position. The performance gap varied primarily across tokenizers rather than masking strategies—the latter largely following patterns already established in property prediction. This analysis therefore focuses on tokenizer differences, using the Combined masking configuration as representative of each tokenizer, since it consistently yields the strongest results. Task+Query is included separately as an additional case, given its anomalous below-random performance, which warrants independent examination, as discussed further in this section.

Unlike property prediction, where ACT must infer masked panel properties from context alone, the classification task analyzed here involves determining properties of visible panels when they occupy the query position during DCM testing. We analyze classification accuracy for both the correct answer panel and the seven incorrect answer panels (which we refer to as "wrong" answers in metrics and discussions). This differs from ACT classification (Table 4.2), which measures accuracy on context panels—the eight visible panels surrounding the query position. During ACT training, the query position contained learnable mask values or correct answer panels (when completing puzzles during random masking). During DCM testing, this position is occupied by each of the eight answer panels—seven of which are incorrect and thus create invalid puzzle contexts outside the training distribution.

Table 5.4 quantifies classification accuracy for answer panels in the query position across different answer panel types, alongside ACT classification accuracy on context panels, prediction accuracy, and final choice-making performance for comparison. If models engage in dual processing (Sec. 4.5.4)—simultaneously reading visible properties and inferring answer properties from context—correct and incorrect answer panels should produce asymmetric classification accuracy, since only the correct panel produces consistent signals from both processes. The asymmetric patterns below test this prediction directly.

Table 5.4: DCM classification performance analysis. *Classification (Answer Panels)* metrics measure classification when answer panels occupy the query position during DCM testing: *Correct* denotes classification accuracy for the correct answer panel; *Incorrect* indicates average classification accuracy across the seven incorrect answer panels; *All* shows overall classification accuracy across all eight answer panels; *WmC* (Wrong matches Correct) represents the percentage of incorrect answer panels misclassified with properties matching the correct answer. *ACT Class.* measures accuracy on the eight *context panels* surrounding the query position. *Corr* shows query panel prediction accuracy (proportion of tasks where all properties are predicted correctly); *PropRate* indicates average per-property prediction accuracy. *ACT Class.*, *Prediction*, and *Choice* columns are included for comparison and taken from Tables 4.2 and 5.2. Note: *Correct* (classification accuracy for the correct answer panel) and *Corr* (proportion of tasks with all predicted properties correct) are distinct metrics despite the similar names.

		Classification (Answer Panels)				ACT Class.	Prediction		Choice
Tokenizer	Masking	Correct	Incorrect	All	WmC	(Context)	Corr	PropRate	
Task	Query	7.52	0.63	1.49	8.06	89.69	18.91	81.23	5.44
	Combined	88.44	74.56	76.30	0.33	88.33	75.63	96.15	96.97
Row	Combined	87.48	54.96	59.02	14.94	87.85	77.58	96.47	82.84
Panel	Combined	68.10	65.87	66.15	0.21	98.28	22.17	83.33	46.85

Degraded Classification Reveals Distribution Shift. Comparing answer panel classification (in the query position) against ACT context panel classification reveals marked degradation across configurations.

Task+Combined shows a moderate drop: 76.30% overall classification of answer panels versus 88.33% on context panels (−12.03 points). Row+Combined exhibits a more pronounced degradation: 59.02% versus 87.85% (−28.83 points). Panel+Combined shows the largest drop: 66.15% versus 98.28% (−32.13 points), likely amplified by the model's weak prediction capabilities confounded with classification at the query position.

Asymmetric Classification Accuracy Reveals Dual Processing. The data expose a consistent pattern: classification accuracy for the correct answer panel exceeds the accuracy for incorrect answer panels across all configurations. Task+Combined achieves 88.44% on correct answers but only 74.56% on incorrect answers (−13.88 points). Row+Combined shows an even more pronounced disparity: 87.48% versus 54.96% (−32.52 points). This asymmetry supports the *dual processing hypothesis* (Sec. 4.5.4), with the accuracy gap quantifying the degree to which conflicting signals from the incorrect panels degrade classification.

Systematic Misclassification Creates Ambiguity. The *WmC* (Wrong matches Correct) metric, defined as the percentage of incorrect answer panels that are misclassified with property vectors matching those of the correct answer, quantifies a particularly damaging error pattern. High WmC values directly undermine DCM's similarity-based decision framework: when multiple answer panels receive property vectors equally similar to the prediction p, the fixed distance metric cannot discriminate between them, leading to incorrect selections even when p is accurate.

Row+Combined exhibits 14.94% WmC, meaning nearly one in seven incorrect answers is misclassified to match the correct answer's properties. While the Row tokenizer's channel-wise panel stacking (Sec. 4.2.2) demonstrably improves property prediction (77.58% Corr, the highest among tokenizers) and achieves strong performance in optimistic estimates (95.90%), this same architectural feature may introduce biases that degrade classification of answer panels. The direct juxtaposition of panels in input channels enables the CNN to capture low-level differences efficiently, enhancing pattern recognition. However, this may also create prediction-oriented biases that interfere with pure property classification when incorrect panels occupy the query position, manifesting in the elevated WmC metric. It is poor incorrect-answer classification and, in particular, the elevated WmC that account for the 13.06-point gap between Row+Combined's choice accuracy and its optimistic estimate, despite strong property prediction. Conversely, Task+Combined's near-balanced classification across correct and incorrect panels, with negligible WmC, explains why it achieves 96.97% choice accuracy slightly exceeding its optimistic estimate, demonstrating that learned representations can surpass ground truth comparisons within DCM's framework. The visualizations presented below provide qualitative evidence for these patterns (see Figure 5.2).

Visualization of Classification Patterns

The classification asymmetries and systematic errors documented in Table 5.4 manifest visually in the model's behavior on individual tasks. Figure 5.2 compares the behavior of Task and Row models on four RAVEN test tasks, illustrating how classification quality affects choice-making even when predictions are accurate.

Example (a) illustrates the issues that affect the Row tokenizer. Both tokenizers produce good results: the Task model identifies all properties perfectly from the query panel as well as answer panels, while the Row tokenizer also achieves good prediction and classification quality. However, the Row model makes two critical mistakes: it misclassifies the *color* property of the correct answer panel (green instead of red for panel p8), while giving an incorrect answer panel (p2) properties that match the target, causing DCM to select p2 as the solution.

Example (b) follows a similar pattern. The Task tokenizer predicts everything perfectly. The Row model makes a good prediction but commits mistakes in classifying the correct answer, making the

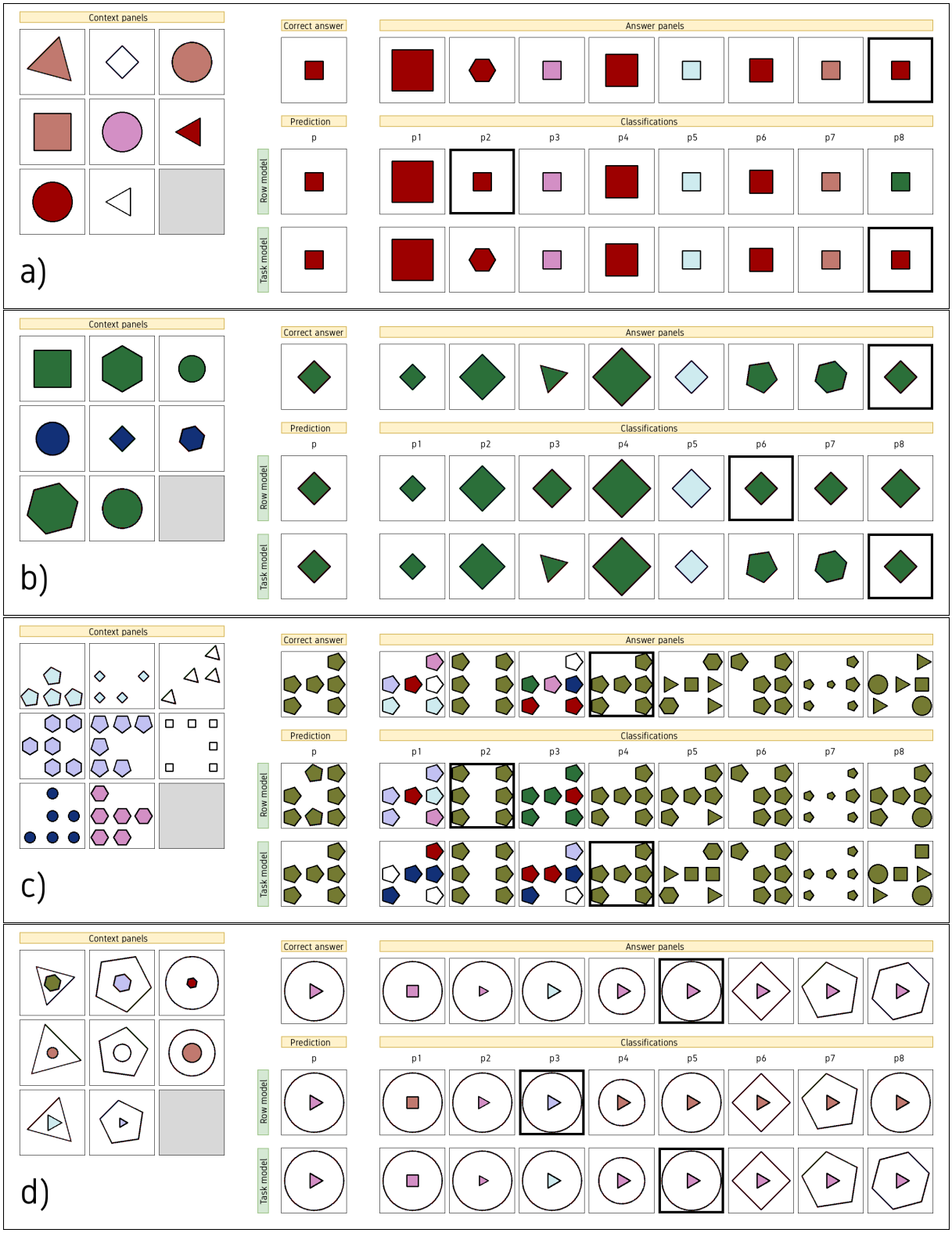


FIGURE 5.2: Solving four RPM tasks (a, b, c, and d) with **Task** and **Row** models (both trained in **Combined** masking mode). **Left:** the task. **Middle:** the correct answer and rendering of models' predictions p (Step 1 of **DCM**) for the Row and Task model. **Right:** answer panels and the renderings of classifications p_i generated by **Row** and **Task** model (Step 2 of **DCM**). The panels corresponding to the most similar property vectors are marked with thicker borders. Predictions and classifications are rendered from property vectors produced by the model while fixing rotation angles, as the angle was irrelevant in those tasks. To facilitate analysis, we render the *color* property using pseudocoloring (it is conventionally rendered in grayscale). Additional examples are provided in Figs. A.1–A.2.

square too large in panel p8. The chosen incorrect answer (panel p6) is classified as if it were correct. One additional incorrect answer (panel p7) receives properties resembling the target, corresponding to the high *WmC* metric (14.94%) for Row+Combined documented in Table 5.4. The dual processing conflict is particularly visible in the *type* property: DCM classifies all answer panels as containing squares (matching the prediction), even though not all answer panels actually contain squares. This systematic bias toward the predicted properties reveals how prediction overrides visible classification.

Example (c) is a harder task with imperfect classifications for both tokenizers. For the Task tokenizer, the classification is good enough for matching the correct answer. The Row tokenizer exhibits an inversion: classification quality is quite good, and the correct answer (panel p4) is classified properly with all properties correctly identified. No incorrect answer classification matches the correct answer. However, the prediction is wrong, causing the model to choose an incorrect answer (panel p2) that is close in terms of similarity.

Despite this inverted failure mode—where prediction rather than classification causes the error—the patterns characteristic of the *dual processing* conflict remain visible. Several incorrect answer classifications (p5 and p8) lie closer to the correct answer than to the incorrect answer being classified, appearing as interpolations between their actual input and the correct answer. This manifests concretely in the *type* classifications: DCM classifies figures as pentagons (matching the correct answer panel p4, where all figures are pentagons) even in panels where pentagons are sparse or absent. Specifically, in panel p5, all but one of six figures are classified as pentagons despite only two actually being pentagons, while in panel p8, all but one figure are classified as pentagons despite the complete absence of pentagons in the input image.

In example (d), the Row tokenizer again makes a good prediction but produces wrong classifications that cause incorrect selection. The model does not exhibit the *WmC* behavior—no incorrect answer exactly matches the properties of the correct answer. However, the dual processing conflict manifests again: panel p2 is positioned, in the property space under metric d, between the incorrect answer being classified and the correct answer (panel p5)—its triangle is classified one grade too large relative to p2 (*small* instead of *tiny*) and one grade too small relative to p5 (*small* instead of *medium*).

Examples (a), (b), and (d) demonstrate the primary failure mode of the Row tokenizer: good prediction undermined by classification errors. This pattern, observable across additional examples in Sec. A.2.1 of the Supplementary Material, provides compelling evidence for the *dual processing hypothesis* (Sec. 4.5.4).

Below-Random Performance of Task+Query Model

One particularly unusual result is the 5.44% AccProb of DCM with Task+Query (Table 5.2), which falls well below the random choice baseline of 12.5% for an 8-class problem. Prediction quality alone does not explain this: while the 18.91% Corr metric is lower than in other configurations, the PropRate of 81.23% (Table 5.4) shows that individual property predictions are more accurate, and the stringent all-or-nothing nature of Corr cannot by itself produce below-random choice accuracy. Classification failure is instead the primary cause: the optimistic estimates experiment, which bypasses classification entirely by substituting ground-truth property vectors, shows Task+Query achieving 65.09% AccProb (Table 5.1)—a 60-point gap that cannot be attributed to prediction. The classification analysis (Table 5.4) corroborates this, revealing near-complete collapse: only 7.52% accuracy on correct answers and 0.63% on incorrect answers, combined with 8.06% WmC. In the following, we examine the architectural and distributional sources of this collapse.

Architectural Sources of Classification Failure. Understanding this classification collapse requires examining how tokenizers (Sec. 4.2) process the query position during training versus testing. During

training, the query panel position contains learnable mask values. During testing with DCM, this position is replaced with the actual answer panels. This replacement affects our tokenizers differently, depending on their architectural characteristics.

The Task tokenizer processes the entire matrix in a single pass, meaning the extractor region analyzing the bottom-right position encounters only mask values during Query masking training. While random masking places real panels in this position, Query masking exclusively trains on the default task configuration where the query panel is always masked. In contrast, the Row tokenizer runs the extractor three times (once per row), so the same region processes real panels in two out of three runs regardless of which panel is masked. The Panel tokenizer analyzes each panel individually, encountering masks in only one of nine runs. Consequently, when DCM replaces the mask with an answer panel, the Task+Query model faces an unprecedented input configuration (essentially an out-of-distribution example), while Row and Panel tokenizers, as well as Task models trained with random masking, have learned to process panels in that position.

Beyond Distribution Shift: Systematic Bias. Training exclusively on masks creates severe overfitting to this configuration, with zero tolerance for real panel content at the query position. When DCM introduces an actual answer panel, the drastically altered activation patterns propagate through the network, corrupting the model's internal representations. Collectively, these factors—overfitting to the masked query position and systematic misclassification at 8.06% WmC—should produce much lower results, possibly near-random performance through confused but unbiased predictions, not systematically worse-than-random accuracy. The 5.44% result (falling 7 percentage points below the 12.5% random baseline) suggests systematic bias: the model actively selects incorrect answers rather than merely failing to recognize patterns.

We hypothesize that the root cause lies in positional encoding: since the mask consistently occupies the query position during training, the mask itself implicitly encodes that location, and the model may never develop a trustworthy positional representation for a real panel appearing there. As a consequence, when DCM substitutes an answer panel, the model may fail to recognize it as the query and instead treat all nine panels as context—then attempt to produce a property vector that differs from all of them, systematically avoiding the correct answer. Supporting evidence for this hypothesis emerges from the classification patterns in Table 5.4: only 7.52% accuracy on correct answer panels and 0.63% on incorrect ones. The model frequently produces property predictions for answer panels that differ from the actual visible properties—as if deliberately avoiding predicting properties that match the input.

Validating H3. The contrast between Task+Query and Task+Combined provides compelling evidence for **H3** (Sec. 1.2): self-supervision is an adequate training regime for building RPM solvers—provided the masking strategy exposes the model to sufficient variation. Task+Query, trained exclusively with query-only masking, collapses to 5.44% choice accuracy because the extractor region processing the query position never encounters real panel content during training. The identical Task tokenizer architecture trained with random masking (Task+Combined) achieves 96.97%, a difference of over 91 percentage points from the same architecture under a different training regime. It even exceeds its optimistic estimate, showing that self-supervised representations can surpass direct ground-truth comparisons within DCM's framework. Notably, random masking's generalization extends beyond direct training exposure: Task+Combined achieves 74.56% classification accuracy on incorrect answer panels despite never seeing incorrect panels during training. This indicates that self-supervision with random masking teaches robust property classification rather than memorization of training configurations, and underscores that self-supervised training is the decisive factor in RPM solving.

5.1.4 Ablation Studies

The following experiments validate the two design choices—the distance metric and the relevance source—introduced in Sec. 5.1.1, and explore whether using a separate ACT instance for prediction and classification improves DCM performance.

Distance Metric and Relevance Source

ACT outputs probability distributions over property values rather than discrete categories, enabling comparison using probabilistic divergence measures. While cross-entropy aligns with ACT's training objective, DCM compares two model outputs rather than output to ground truth, making it worthwhile to verify whether alternative divergence measures perform better. We evaluated the following:

- Cross-entropy — used as DCM's distance function (Sec. 5.1.1)
- Kullback–Leibler divergence (Kullback and Leibler, 1951)
- Jensen–Shannon divergence (Lin, 1991)
- Their symmetrical versions (averaging $d(p, p_i)$ and $d(p_i, p)$ when not inherently symmetrical)

Each metric was evaluated under both classification- and prediction-derived relevance, yielding the results in Table 5.5.

Table 5.5: DCM accuracy under different divergence metrics, symmetry configurations, and relevance sources. All experiments use the Task tokenizer with combined masking strategy.

Source of relevance	Metric	Symmetric	Accuracy (%)
Classification	Cross-entropy	No	96.97
Classification	Cross-entropy	Yes	96.38
Classification	JS divergence	Yes	95.60
Classification	KL divergence	No	96.63
Classification	KL divergence	Yes	96.21
Prediction	Cross-entropy	No	94.66
Prediction	Cross-entropy	Yes	93.97
Prediction	JS divergence	Yes	93.69
Prediction	KL divergence	No	94.68
Prediction	KL divergence	Yes	93.88

Cross-entropy achieves the best performance (96.97%), confirming its choice as the distance function for DCM. KL divergence performs comparably (96.63%), while JS divergence yields slightly lower accuracy (95.60%). Symmetrization consistently reduces performance across all metrics, confirming that the asymmetric formulation better captures the directional structure of the prediction-to-classification comparison.

The results further confirm the choice of classification as the source of relevance. Using p_i to determine relevant properties outperforms prediction-derived relevance by approximately 2 percentage points across all metrics (96.97% vs. 94.66% for cross-entropy), corroborating both reasons stated in Sec. 5.1.1: classification of a visible panel is more accurate than prediction, and target-derived relevance focuses each comparison on properties pertinent to that specific answer panel.

5.1.5 Decoupled Prediction and Classification Weights

Dual-network architectures in **deep reinforcement learning** (Double DQN (van Hasselt et al., 2016), Dueling DQN (Wang et al., 2016)) and **contrastive learning** (MoCo (He et al., 2020), SimCLR (Chen

et al., 2020b), BYOL (Grill et al., 2020)) suggest a natural extension of DCM: using separate weights for prediction and classification, with dedicated networks trained jointly via momentum-based cross-network updates. This would align closely with the *dual processing hypothesis* (Sec. 4.5.4), explicitly separating the two pathways while facilitating knowledge transfer between them. Before committing to the substantial engineering effort such an architecture requires, we first evaluated whether the high-level idea—using separate weights for prediction and classification—showed any promise. We did so using a simpler proxy: weights from different ACT training phases (Random and Combined masking) assigned independently to DCM's prediction and classification. In the standard configuration, both use weights from the same training phase; here we mix them so that weights from one phase handle prediction while weights from the other handle classification. We tested two assignments for each tokenizer:

- **Combined / Random**: weights from the Combined masking phase for prediction; weights from the Random masking phase for classification. The rationale was that the Combined model, having undergone query-specific fine-tuning, might produce superior predictions for the query panel, while the Random model, trained across all panel positions, might provide more dependable classification of answer panels.
- **Random / Combined**: the reverse assignment, tested for completeness, though we had no strong expectation that Random weights for prediction would outperform the Combined model.

Table 5.6 presents AccProb for these configurations, compared against single-model baselines from Table 5.2.

Table 5.6: Performance of DCM when using weights from different training phases for prediction and classification. **Prediction** indicates which model's weights are used for query panel prediction. **Classification** indicates the weights used for answer panel classification. Baseline configurations use the same weights for both (from Table 5.2).

Tokenizer	Prediction	Classification	AccProb (%)
Panel	Random	Random	41.39
	Combined	Combined	46.85
	Combined	Random	45.33
	Random	Combined	42.42
Task	Random	Random	96.33
	Combined	Combined	96.97
	Combined	Random	96.66
	Random	Combined	96.88
Row	Random	Random	79.23
	Combined	Combined	82.84
	Combined	Random	82.79
	Random	Combined	79.11

Contrary to the initial hypothesis, mixing weights from different training phases consistently degraded or failed to improve performance across all tokenizer configurations. The Combined / Random configuration showed decreases ranging from 0.31 percentage points (Task: 96.66% vs. 96.97%) to 1.52 points (Panel: 45.33% vs. 46.85%), while remaining at or below the Random baseline. The Random / Combined configuration performed worse, with degradation ranging from 0.09 points (Task: 96.88% vs. 96.97%) to 4.43 points (Panel: 42.42% vs. 46.85%), with performance generally approximating the Random baseline. The Row tokenizer exhibited particularly pronounced asymmetry: Combined / Random maintained near-baseline performance (82.79% vs. 82.84%), while Random / Combined dropped to 79.11%—nearly matching the Random baseline of 79.23%.

We attribute these results to **distributional mismatch**: although both models share identical architectures, their weights come from different training phases—one trained with Random masking and one

with Combined masking—causing the probability distributions they produce over the same properties to be insufficiently calibrated against each other for stable cross-entropy comparison. These findings indicate that representational consistency matters more than specialized weights for DCM's performance, and suggest that more elaborate dual-network architectures with momentum-based updates would face similar or greater compatibility challenges.

5.1.6 Limitations of DCM

Despite its strengths, DCM has several inherent limitations that motivate the development of more sophisticated approaches.

Fixed distance function. DCM employs a fixed cross-entropy distance to compare the predicted properties p with the classified properties p_i of answer panels. This rigid similarity metric cannot adapt to varying classification quality or account for the systematic errors documented in Sec. 5.1.3. When classification degrades for incorrect answer panels—as demonstrated by Row+Combined's 14.94% WmC metric (Table 5.4)—the fixed distance function cannot distinguish true matches from false positives created by misclassification.

Impact of the source of relevance. Computing cross-entropy between p and p_i requires designating one as the source of relevance to determine which properties matter for comparison. DCM uses p_i for this role (Sec. 5.1.1); consequently, any error in p_i causes similarities to be assessed on incorrect properties, amplifying the impact of classification errors.

Direct comparison assumption. This constraint leads to a deeper issue: DCM assumes that the predicted properties p (from a masked query panel) can be directly compared with the classified properties p_i (from answer panels placed in context). While ACT is trained to produce properties of the masked query panel, obtaining reliable p_is for incorrect answer panels proves problematic. As demonstrated in Sec. 5.1.3, classification performance differs substantially between correct and incorrect answer panels. This assumption's validity depends on understanding how ACT processes answer panels when they occupy the query position—a question that exposes limitations in both the current approach and its most natural alternative. In a sense, DCM's pair-wise selection of the answer panel most similar to p constitutes a form of instance-based reasoning in property space, akin to nearest-neighbor retrieval (Cover and Hart, 1967)—a design that succeeds when p_is are well-calibrated but fragile when classification degrades.

An Alternative Framework for Answer Panel Evaluation. The direct comparison assumption rests on a specific view of how ACT processes answer panels. DCM currently employs the first framework, which treats the model as an accurate classifier of visible properties in answer panels, using p_i as faithful representations. However, as discussed in Sec. 4.5.4, models engage in *dual processing*; simultaneously performing classification (reading visible properties) and prediction (inferring answer properties from context). When an incorrect answer panel occupies the query position, it creates an invalid context outside the training distribution. The model may continue attempting to predict correct answer properties even when confronted with a visible but incorrect panel, degrading p_i accuracy for incorrect answers. Sec. 5.1.3 empirically demonstrated this effect: classification accuracy for incorrect answer panels consistently falls below that for correct answers (Task+Combined: 88.44% correct vs. 74.56% incorrect; Row+Combined: 87.48% vs. 54.96% — Table 5.4).

Recognizing this, one could consider reformulating DCM around a **second framework** that attempts to exploit this dual processing behavior rather than fight it. When a correct answer occupies the query position, its visible properties align with the context-derived prediction—classification and prediction cooperate, yielding a consistent representation. When an incorrect answer is placed there instead, the visible properties conflict with what the context predicts, potentially producing a detectable mismatch. At first glance, this resembles *autoencoder*-based anomaly detection (An and Cho, 2015; Hendrycks and

Gimpel, 2016): samples drawn from the training distribution are faithfully reconstructed, while *out-of-distribution* inputs produce elevated reconstruction errors (Hendrycks and Gimpel, 2016). Analogously, incorrect answer panels might generate prediction-classification mismatches that signal their incorrectness.

However, this second framework encounters an unavoidable obstacle: measuring such a mismatch requires ground truth properties for each answer panel, or at minimum a sound estimate thereof. In autoencoder anomaly detection, reconstruction error is computed against the known input; here, we would need true panel properties to quantify how far the classified p_i deviates from what was predicted—information that is unavailable at inference time. Both frameworks therefore face their own irresolvable issues within the current ACT design: the former because the model performs dual processing rather than pure classification, and the latter because neither ground truth nor a valid estimate of answer panel properties is available at inference time.

Why Not Use a Separate Classification Network? An isolated classification model could in principle resolve both issues—providing pure classification for the former framework and ground truth estimates for the latter. However, as shown in Sec. 5.1.5, even models with identical architectures differing only in training phase produce distributions insufficiently calibrated against each other for stable cross-entropy comparison. A structurally distinct classifier would face greater incompatibility still, and would additionally lack the context-derived relevance assessments that ACT's outputs carry.

Given these considerations, we improve DCM by extending the existing framework rather than restructuring it with a separate classification model. The **Learnable Choice Maker (LCM)**, presented in the following section, addresses these limitations by learning adaptive similarity functions that can account for varying classification quality, adjust comparisons based on prediction confidence, and implicitly handle the dual processing behavior that undermines DCM's direct comparison assumption.

5.2 Learnable Choice Maker (LCM)

Building on the limitations identified in Sec. 5.1.6, we introduce the **Learnable Choice Maker (LCM)**—a learned similarity function that addresses them. Rather than relying on handcrafted cross-entropy comparisons, LCM learns a parameterized similarity function capable of discerning the relative importance of different properties across answer panel comparisons.

5.2.1 Method

Architecture

LCM employs a five-layer *multilayer perceptron* (MLP) applied to pairs of latent vectors. Specifically, it processes concatenated log-probability vectors z and z_i rather than the final activated property vectors p and p_i; z and z_i are obtained by removing the activation function from the last ACT layer, resulting in 557-dimensional vectors (Sec. 4.2.4). The MLP receives $2 \times 557 = 1114$-dimensional input vectors and comprises four hidden fully connected layers with 1024 neurons each, using GELU activation functions and layer normalization ($\epsilon = 0.001$) before each layer. The output layer consists of a single neuron producing a similarity score. The MLP is queried separately for each of the eight candidate answer panels; the resulting scores are passed through a softmax function and optimized using categorical cross-entropy loss with the true answer position encoded as a one-hot vector. During inference, the candidate panel with the highest score is selected. Detailed architecture specifications, tensor flow diagrams, and computational resource requirements are provided in Appendix A.1.

Training Scenarios

We consider three training scenarios for combining LCM with ACT models:

- **ACT+LCM**: A trained ACT model is combined with LCM and the entire system is trained on the RAVEN training set until convergence, allowing ACT parameters to undergo further updates.
- **ACT$_f$+LCM**: The procedure is similar to ACT+LCM, but ACT parameters are frozen, and only LCM is trained.
- **Fresh**: The entire ACT+LCM architecture is randomly initialized and trained end-to-end from scratch without property prediction masking constraints.

Note that in the ACT+LCM and Fresh training regimes, the property vectors returned by ACT become latent variables and should not be expected to explicitly encode panel properties as mandated by the ACT training regime. In these scenarios, LCM learns similarity comparisons in a potentially transformed latent space rather than the original property space.

Training Setup

All three scenarios were trained using the Adam optimizer with a learning rate of 3.5×10^{-5} and batch size of 32. This relatively small learning rate compared to ACT training (0.001) reflects the fine-tuning nature of the choice-making stage. Training proceeded for up to 200 epochs with early stopping based on validation performance to prevent overfitting. We used the RAVEN training set (56,000 tasks; Sec. 3.1.1) with early stopping based on the RAVEN validation set (14,000 tasks). All hyperparameters were determined through preliminary experiments and remained fixed across scenarios.

5.2.2 Experimental Results

Table 5.7: Accuracy of LCM models on choice tasks using three training procedures: ACT+LCM (trained property predictor subsequently trained together with LCM); ACT$_f$+LCM (trained property predictor is frozen and only LCM is trained subsequently); Fresh (the entire ACT+LCM combination is trained from scratch without masking).

		RAVEN			I-RAVEN			RAVEN-FAIR		
Tokenizer	Masking	ACT+LCM	ACT$_f$+LCM	Fresh	ACT+LCM	ACT$_f$+LCM	Fresh	ACT+LCM	ACT$_f$+LCM	Fresh
Task	Random	97.24	94.67	28.34	96.59	94.43	50.24	99.05	98.21	64.41
	Combined	97.44	95.86		97.14	95.15		99.10	98.56	
Row	Random	97.29	90.86	34.08	93.37	89.06	55.40	98.93	96.85	67.71
	Combined	97.42	92.21		93.26	89.70		98.88	97.36	

Table 5.7 presents the percentage of tasks solved by ACT models combined with LCM in the three training scenarios, evaluated on the RAVEN (Sec. 3.1.1), I-RAVEN (Sec. 3.2.3), and RAVEN-FAIR (Sec. 3.2.3) benchmarks.

On the RAVEN and I-RAVEN benchmarks, ACT+LCM systematically exceeds DCM performance by 1–2 percentage points (cf. Table 5.2), corroborating the anticipated benefits of adaptive similarity functions. The vectors produced by ACT in two querying modes (classification and prediction) may have different characteristics that a fixed distance function cannot fully capture. LCM adapts to those differences, in effect learning a data-driven similarity metric.

Notably, Row tokenizer models perform comparably to Task tokenizer models under LCM, whereas they underperformed markedly with DCM (82.84% vs. 96.97%). This indicates that LCM successfully compensates for ACT deficiencies through joint fine-tuning.

When ACT is frozen during LCM training (ACT$_f$+LCM), resulting models perform worse (Task+Combined: 95.86% vs. 97.44%; Row+Combined: 92.21% vs. 97.42%), indicating that inferring high-accuracy decisions from fixed representations proves more difficult than allowing joint adaptation. The poor performance of models trained from scratch (**Fresh**: 28.34–34.08%) confirms that *self-supervised* ACT pretraining is essential, providing evidence for **H3** (Sec. 1.2), examined in depth in Sec. 5.2.7.

5.2.3 Classification Performance Analysis

Although prediction and classification use the same predictor model, they operate on different inputs: prediction targets the masked query position, while classification processes each answer panel placed in that position. The DCM analysis in Sec. 5.1.3 showed that placing incorrect answer panels in the query position creates systematic classification asymmetries—with correct panels classified reliably and incorrect ones much less so. We now examine whether LCM's joint training on the choice-making objective alters these patterns and what the consequences are for final task-solving performance. LCM improves incorrect answer panel classification relative to DCM across all tokenizers and nearly eliminates systematic misclassification (WmC), despite being trained exclusively for choice-making rather than property identification.

Table 5.8 presents classification and choice-making performance on the RAVEN test set for the Task and Row tokenizers under Combined masking, comparing DCM and LCM directly.

Table 5.8: Classification performance on RAVEN across tokenizers and choice makers (Combined masking). *Correct* denotes accuracy for the correct answer panel; *Incorrect* shows average accuracy across incorrect answer panels; *All* indicates overall accuracy; *WmC* represents the Wrong matches Correct metric; *Prediction* shows query panel prediction accuracy (proportion of tasks where all properties are correct); *Choice* indicates final task-solving accuracy.

		Classification				Prediction	Choice
Tokenizer	Type	Correct	Incorrect	All	WmC		
Task	DCM	88.44	74.56	76.30	0.33	75.63	96.97
	LCM	68.28	55.82	57.34	2.86	68.28	97.44
Row	DCM	87.48	54.96	59.02	14.94	77.58	82.84
	LCM	73.09	67.23	68.80	0.39	73.09	97.42

The Row+Combined rows show this trade-off most sharply: a high rate of correct answer classification at the cost of systematic misclassification of incorrect panels. As documented in Sec. 5.1.3, DCM excels at classifying the correct answer (87.48%) but degrades on incorrect answers (54.96%), with 14.94% of them misclassified to match the correct answer's properties—an ambiguity that directly undermines the similarity-based decision framework. LCM reverses this pattern: correct answer classification is lower (73.09%), but incorrect answer classification improves to 67.23%, and WmC falls to near-zero (0.39%). The consequence for choice-making is decisive: despite DCM's higher prediction accuracy (77.58% vs. 73.09%), LCM reaches 97.42% versus DCM's 82.84%. Classification rate for incorrect answer panels proves more decisive than prediction accuracy or correct answer classification rate alone.

An additional notable finding concerns tokenizer performance: while the Task tokenizer still outperforms the Row tokenizer for LCM, the gap narrows considerably compared to property prediction alone, suggesting that joint training on the selection task enhances the model's ability to classify properties—examined in depth in Sec. 5.2.5.

Conditional Analysis: How Classification Quality Affects Choice Making

To understand how classification performance interacts with prediction quality, Tables 5.9 and 5.10 present task-solving accuracy conditioned on two factors: prediction correctness (whether all relevant

properties of the query panel were correctly predicted) and correct answer classification rate (whether the correct answer panel's properties were correctly classified when occupying the query position). We use *Exact* to denote that all properties of a given panel were identified without error—an exact match between predicted and ground-truth properties—and *Inexact* to denote that at least one property was misidentified. These labels describe model accuracy on a panel and are independent of whether that panel is the correct answer panel or one of the incorrect answer panels: a correct answer panel may receive an Inexact rating if the model misidentifies any of its properties, and an incorrect answer panel may receive an Exact rating if all its properties are correctly identified. Columns represent prediction correctness, rows represent correct answer classification rate, and cell values show task-solving accuracy (%). Parenthetical percentages indicate the share of the test set in each category.

Table 5.9: DCM accuracy conditioned on prediction and correct answer classification (Row tokenizer, Combined masking).

	Prediction	
Correct Answer Classification	Inexact (21.93%)	Exact (78.07%)
Inexact (12.52%)	67.57 (7.91%)	71.52 (4.61%)
Exact (87.48%)	82.53 (14.02%)	85.87 (73.46%)

Table 5.10: LCM accuracy conditioned on prediction and correct answer classification (Row tokenizer, Combined masking).

	Prediction	
Correct Answer Classification	Inexact (26.91%)	Exact (73.09%)
Inexact (20.19%)	89.08 (11.39%)	96.03 (8.81%)
Exact (79.81%)	93.42 (15.53%)	99.16 (64.28%)

For DCM (Table 5.9), correct answer classification rate is the stronger performance driver. When it is Exact, DCM achieves 82.53–85.87% regardless of prediction outcome; when it is Inexact, performance falls to 67.57–71.52%. Prediction failure alone (Inexact prediction, Exact classification; 14.02% of cases) reduces accuracy only modestly to 82.53%, indicating that DCM partially compensates for imperfect predictions when the correct answer is well classified. The more damaging condition is correct answer misclassification, which reduces accuracy to 71.52% even when prediction is Exact (top-right cell; 4.61% of cases).

For LCM (Table 5.10), the pattern differs markedly. LCM maintains consistently high accuracy (89–99%) across all four combinations. When both prediction and correct answer classification are Exact (bottom-right), LCM achieves near-perfect 99.16% accuracy in 64.28% of cases. When prediction is Inexact but correct answer classification is Exact (bottom-left; 15.53% of cases), LCM maintains 93.42%—10.89 percentage points above DCM's 82.53% in the same condition. More strikingly, even when correct answer classification is Inexact, LCM achieves 89.08% and 96.03%, far above DCM's 67.57% and 71.52% in the equivalent combinations. LCM's learned similarity functions thus compensate for imperfect classification more robustly than DCM's fixed metric across every combination.

The Role of Incorrect Answer Classification

However, Tables 5.9 and 5.10 focus solely on prediction correctness and correct answer classification rate, omitting incorrect answer panel classification—a critical factor for distinguishing answer candidates.

Tables 5.11 and 5.12 extend the conditional analysis by tracking classification correctness for both the correct answer panel and the seven incorrect answer panels. The column structure is identical to the previous tables (Prediction: Inexact/Exact). Rows now split cases along two dimensions—correct answer classification rate and incorrect answer classification rate—each rated Exact or Inexact, yielding four row

combinations. Cell values show task-solving accuracy (%), with parenthetical percentages indicating the share of the test set in each combination.

Note that *Exact* for incorrect answer panels requires all properties across all seven panels to be correctly identified—a stricter criterion than *Exact* for the correct answer panel, which covers a single panel. Consequently, *Inexact* for incorrect answer panels does not indicate total failure but rather at least one property error across the seven panels, even if minor (e.g., a single slightly incorrect size value). Many such cases involve otherwise correct classifications with only isolated errors.

Table 5.11: DCM accuracy with detailed classification conditioning (Row tokenizer, Combined masking).

Classification		Prediction	
Correct Answer Panel	Incorrect Answer Panels	Inexact (21.93%)	Exact (78.07%)
Inexact (12.52%)	Inexact (89.99%)	67.57 (7.91%)	71.38 (4.59%)
	Exact (10.01%)	— (0.00%)	100.00 (0.02%)
Exact (87.48%)	Inexact (89.99%)	82.42 (13.69%)	83.73 (63.80%)
	Exact (10.01%)	86.96 (0.33%)	100.00 (9.66%)

Table 5.12: LCM accuracy with detailed classification conditioning (Row tokenizer, Combined masking).

Classification		Prediction	
Correct Answer Panel	Incorrect Answer Panels	Inexact (26.91%)	Exact (73.09%)
Inexact (20.19%)	Inexact (56.66%)	89.11 (11.35%)	95.97 (8.69%)
	Exact (43.34%)	80.00 (0.04%)	100.00 (0.11%)
Exact (79.81%)	Inexact (56.66%)	93.81 (12.91%)	98.64 (23.71%)
	Exact (43.34%)	91.53 (2.61%)	99.45 (40.57%)

For DCM (Table 5.11), the simplified analysis suggested that correct answer classification rate may outweigh prediction quality as a performance driver. The detailed breakdown indicates that incorrect answer classification rate is likely the most decisive of the three factors, though the interactions are intricate and some subcases cover only a small fraction of the test set, making it difficult to draw firm conclusions about their relative importance. The key finding concerns the row marginals: correct classification of all properties across all seven incorrect answer panels (Exact rows for incorrect answers) occurs in only 10.01% of cases. In this ideal combination, when prediction succeeds as well, DCM reaches 100% regardless of correct answer classification rate (though the case where correct answer classification is Inexact covers only 0.02% of the test set). When incorrect answer classification is Exact but prediction fails, accuracy drops to 86.96%, suggesting that while prediction may be the weakest of the three drivers, all three contribute meaningfully to performance. In the remaining 89.99% of cases, at least one classification error across the incorrect panels is present, and performance drops markedly to 67–84%, depending on prediction and correct answer classification rate. This severely limits DCM's ability to distinguish answer candidates through direct property comparison.

For LCM (Table 5.12), 43.34% of cases involve correct classification of all properties across incorrect answer panels, compared to only 10.01% for DCM—a 33-percentage-point improvement. When all panels are correctly classified and prediction succeeds (bottom-right cell), accuracy reaches 99.45% in 40.57% of cases, compared to DCM's 100% in only 9.66% of cases. The near-identical peak accuracy (99.45% vs. 100%) combined with substantially higher coverage (40.57% vs. 9.66%) underscores LCM's advantage.

These detailed tables reveal why the earlier analysis (Tables 5.9 and 5.10) could be misleading. Based on those tables alone, DCM may appear superior: higher prediction accuracy (78.07% vs. 73.09%), higher correct answer classification rate (87.48% vs. 79.81%), and more cases where both succeed simultaneously (73.46% vs. 64.28%). However, this advantage is illusory because those tables ignore classification rate for incorrect answer panels—the decisive factor for candidate discrimination.

LCM's robustness is further evident in the worst-case combination: when prediction, correct answer classification, and incorrect answer classification all contain at least one error (top-left cell), LCM still achieves 89.11%, well above DCM's 67.57%. The learned comparator remains efficient even when property identification is imperfect across all three dimensions.

5.2.4 Visual Evidence

Figure 5.3 juxtaposes ACT+DCM and ACT+LCM behavior (both using Row tokenizer) on four representative tasks, providing qualitative support for the quantitative findings. Examples (b) and (c) correspond to examples (a) and (b) from the Row versus Task tokenizer comparison in Figure 5.2, with the first row in both figures showing DCM with Row tokenizer to facilitate direct cross-comparison.

In example (a), both DCM and LCM perfectly predicted the correct answer panel. However, DCM committed multiple classification errors on answer panels, particularly on the *type* attribute. These classification errors caused DCM to identify panel p6 rather than the correct p8 as the best match. In contrast, LCM's superior classification of incorrect answer panels enabled it to correctly distinguish between candidates.

Example (b) demonstrates LCM's ability to recover from prediction errors through superior classification. Neither model produced a perfect prediction, but LCM made an error at only one of nine object locations, while DCM made three such mistakes. Combined with perfect classification of the correct answer (panel p4), LCM selected the right answer despite imperfect prediction. Interestingly, LCM's classification of p2 diverges from the prediction by just one object—qualitatively similar to the divergence for p4—yet LCM correctly chose p4, likely benefiting from continuous probability distributions rather than discrete property assignments.

Example (c) presents a particularly revealing scenario: LCM selected the correct answer (panel p4) even though both its prediction and classification were imprecise. Despite these dual imperfections, LCM's learned comparator function successfully identified p4 as the best match, demonstrating robustness to correlated errors that DCM's fixed distance metric cannot provide.

Example (d) reveals another interesting failure mode of DCM. The model makes an incorrect choice, as multiple incorrect answers visually resemble the correct answer. The prediction is not fully correct, resulting in multiple answers that match the correct panel but not the prediction. LCM operates with an identically imperfect prediction. While classification of the correct answer is good, the overall classification is not accurate, with incorrect answers classified at a similar level to DCM or worse. However, LCM makes the correct choice through its trained comparator function: the learned comparison function better fits the task and incorporates context, and LCM's classification errors differ from DCM's systematic tendency to assign incorrect answers properties that match the target—the pattern that directly undermines DCM's fixed similarity metric.

The visualizations confirm that the predictions and classifications produced by ACT+LCM remain interpretable and closely match actual panel properties—noticeably better than DCM. Even when LCM makes classification errors, these mistakes do not exhibit the systematic *WmC* bias where incorrect answers receive properties matching the target, a pattern that stems from the *dual processing* conflict in DCM (Sec. 5.1.3), demonstrating that joint optimization enables the model to disentangle prediction and classification processes while maintaining interpretable property-level representations. Additional examples are provided in Sec. A.2.1 of the Supplementary Material.

5.2.5 Emergent Improvements from Choice-Making Training

The improvement in incorrect answer classification is notable because LCM achieved this while learning an entirely different task. As described in Sec. 5.2, LCM receives only the index of the correct answer

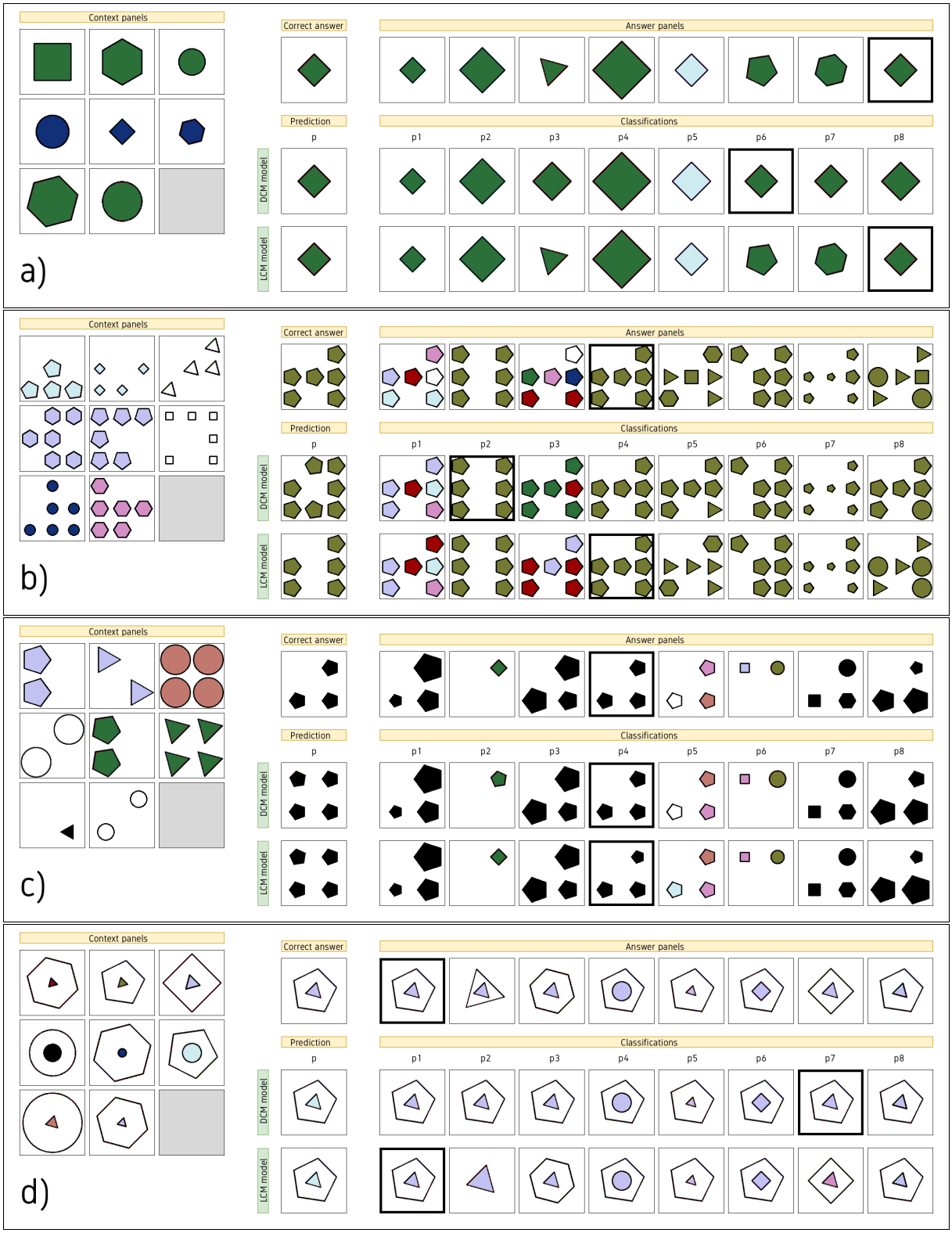


FIGURE 5.3: Behavior of ACT+DCM and ACT+LCM on four RPM tasks from the test set. Each example shows the task (left), the correct answer panel (center top), and all answer panels (top right). Below them, we visualize (independently for DCM and LCM) the prediction for the query panel (middle) and the classification of the answer panels (right), with the one indicated by the choice maker marked by a thick border. The first row of classification results shows DCM with Row tokenizer; the second row shows LCM with Row tokenizer. In Figure 5.2, the first row likewise shows DCM with Row tokenizer, while the second shows DCM with Task tokenizer, enabling direct cross-visualization comparison. When rendering answer panels, rotation angles are fixed, as they are irrelevant in these tasks. For better visibility, we render the *color* property using pseudocoloring rather than grayscale. Additional examples are provided in Sec. A.2.1 of the Supplementary Material.

panel (1–8) as supervision—it is not explicitly trained to predict or classify panel properties, yet through the choice-making objective it substantially improved ACT's ability to classify incorrect answer panels.

This emergent capability demonstrates that *self-supervised* property prediction establishes foundational representations that subsequent task-specific training refines rather than replaces. If the initial representations were shallow or task-specific, LCM training would likely discard them in favor of shortcuts optimized solely for answer selection. Instead, LCM builds upon and refines these representations, providing further evidence for **H3** (Sec. 1.2).

Moreover, LCM training narrows performance gaps between tokenization schemes. While the Task tokenizer still outperforms Row, the gap decreases substantially compared to property prediction alone (Table 4.2 vs. Table 5.7). This suggests that joint optimization with the choice-making objective addresses the classification shortcomings of DCM identified in Sec. 5.1.3, even without explicit supervision on panel properties across both tokenizers.

5.2.6 Data Scalability Analysis

The two-stage design of ACT—which learns property prediction through self-supervision before being combined with LCM (Sec. 4.1)—suggests that our approach may scale more efficiently with respect to available training data. We hypothesize this advantage arises because ACT is trained to solve a challenging multi-label classification task, predicting values for 101 attributes across 9 panels (Sec. 4.2.5). This rich supervisory signal constrains the space of feasible model parameterizations far more than directly selecting among 8 answer panels, making overfitting less likely.

To evaluate this data efficiency hypothesis, we prepared reduced training sets by sampling 5%, 25%, and 50% of examples from the original RAVEN training set (Sec. 3.1.1). We trained the ACT+LCM architecture with Task tokenizer in Combined masking mode (the best-performing configuration from Table 5.7) under three scenarios:

- **R+F**: Training ACT on the *reduced* training set, followed by training ACT+LCM on the *full* training set
- **F+R**: Training ACT on the *full* training set, followed by training ACT+LCM on the *reduced* training set
- **R+R**: Training ACT on the *reduced* training set, followed by training ACT+LCM on the *reduced* training set

All other hyperparameters remained identical to previous experiments. Each data point in Figure 5.4 represents an independent two-stage training process of ACT followed by ACT+LCM. For reference, the rightmost point corresponds to the model trained entirely on the full training set (second row of Table 5.7).

Results and Interpretation

The **F+R** variant demonstrates robustness, experiencing minimal performance degradation across all reduced dataset sizes. When LCM training uses only 50% of data, accuracy remains near the 97.44% achieved with full data. Even with only 25% or 5% of data for LCM training, F+R maintains high performance. This modest decline likely stems mainly from the reduced number of weight updates overall: with fewer examples per epoch, the same number of epochs yields shorter training duration than on the full set. With extended training schedules, we could likely reduce this performance gap further.

In contrast, the **R+F** and **R+R** variants show substantially greater sensitivity to reduced ACT training data, though they still maintain approximately 90% accuracy at 50% data and degrade more

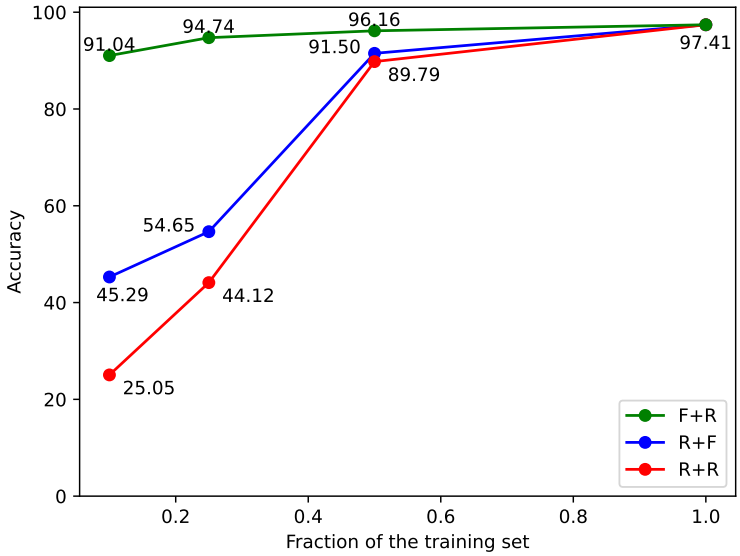


FIGURE 5.4: Test-set accuracy of the ACT+LCM architecture trained on reduced training data

gracefully than training from scratch. Notably, even R+F and R+R with severely reduced data (5%) perform better than or on par with ACT+LCM trained from scratch on the entire dataset (28.34%, *Fresh* in Table 5.7), demonstrating that the staged training approach provides clear advantages regardless of data availability. The contrast between F+R robustness and R+F/R+R sensitivity provides additional evidence for **H3** (Sec. 1.2), discussed further in Section 5.2.7.

5.2.7 Hypothesis Validation

The results presented across this section—spanning quantitative classification analysis, qualitative visual evidence, emergent improvements from joint training, and data scalability experiments—collectively provide strong validation for our hypotheses (Sec. 1.2). Each piece of evidence contributes a distinct perspective on why the staged, self-supervised framework succeeds where simpler alternatives fail.

Validation of H1: Staged Decomposition Facilitates Better Solvers. By separating property prediction and choice-making, each component can specialize. LCM adopts ACT's representations for choice-making, improving incorrect answer classification (54.96%→67.23%, Sec. 5.2.3) and reducing systematic misclassification (14.94%→0.39%). This specialization is harder to achieve in end-to-end architectures where representations must simultaneously serve both objectives. The staged approach enables targeted optimization: ACT focuses on property prediction, while LCM refines these representations for answer discrimination.

Preliminary experiments with alternative architectures further support this. Variants that gave a single module access to all eight answer panels simultaneously—including models processing all ACT outputs jointly and architectures receiving all answer images at the start—consistently performed worse than the simple pairwise module. This suggests that isolated per-candidate comparison, as enforced by the staged decomposition, is not merely a simplifying design choice but a well-motivated inductive bias for answer selection.

Partial Validation of H2: Property Prediction Mitigates Biases. If property prediction failed to establish bias-resistant foundations, we would expect LCM to discard these representations and learn shortcuts exploiting answer set patterns. Instead, LCM improves classification by refining property

prediction representations. The near-elimination of systematic misclassification (Sec. 5.2.3) confirms that LCM learns to discriminate based on actual properties rather than dataset artifacts, validating our architectural decision to pretrain through property prediction before fine-tuning.

Validation of H3: Self-Supervision Enables Adequate Training. H3 (Sec. 1.2) **is validated.** The pretraining of self-supervised property prediction—through exposure to diverse masking contexts—allows ACT to learn general rules governing RPM structure, establishing representations robust enough so that choice-making training refines rather than replaces them. LCM achieves better overall classification by adopting these representations (Sec. 5.2.3), demonstrating that this training regime creates a strong foundation for downstream reasoning tasks. The poor performance of Fresh (28–34%) versus ACT+LCM (97%+) provides direct empirical confirmation. The data scalability results (Sec. 5.2.6) reinforce this further: once ACT has learned robust property predictions on sufficient data, the choice-making module requires comparatively little data to learn reliable comparison strategies. This pattern is consistent with self-supervision doing most of the heavy lifting, leaving only the final discrimination step for LCM.

These findings suggest that refining the existing ACT model through joint training (Sec. 5.1.6) is more productive than creating a separate classification model. A separate classifier would likely produce incompatible probability distributions, as the experiments in Sec. 5.1.5 indicate. LCM's ability to improve classification while training on a different objective provides evidence that joint training captures synergies that are difficult to achieve in separated architectures.

5.3 Contrastive Choice Maker (CCM)

In contrast to attaching an explicit learnable LCM module to ACT (proposed in the previous section), here we propose a method that continues training (fine-tuning) a pretrained ACT model in a solver mode using a different loss function. This approach, which we term the **Contrastive Choice Maker (CCM)**, employs *contrastive learning* to reward correct choices by positioning the correct answer's representation close to the query prediction in the latent space, while pushing the incorrect answers farther away. The method focuses on the relative interplay of model outputs for decision classes, leveraging geometric relationships rather than comparison functions learned by LCM. Unlike LCM (Sec. 5.2.1), which introduces additional architectural components, CCM operates by directly adapting the existing ACT weights through exposure to supervised choice-making examples, maintaining the architectural simplicity of the property prediction stage while specializing the model for answer selection.

5.3.1 Method

Contrastive Learning Framework

We devise a contrastive learning scheme operating on the same logits vectors z and z_i (pre-activation property predictions) that DCM uses for distance comparison (Sec. 5.1.1), replacing the fixed cross-entropy metric with a learned contrastive objective. The loss function is based on *Normalized Temperature-scaled Cross Entropy (NT-Xent)*, as termed in (Chen et al., 2020b), building on the InfoNCE loss (van den Oord et al., 2018), non-parametric instance discrimination (Wu et al., 2018), and the multi-class N-pair loss (Sohn, 2016).

Let z be ACT's predicted logits vector for the masked query panel (referred to as prediction in the previous sections), and z_i be the logits for the i-th candidate panel (the classifications). In contrastive learning terminology, the correct answer panel forms a *positive pair* with the query prediction, while the seven incorrect answer panels form *negative pairs*; the loss pulls positive pairs together and pushes

negative pairs apart in the representation space. We define the contrastive loss as:

$$L_c(z, z_1, \ldots, z_8) = -\log \frac{\exp(s(z_j, z)/\tau)}{\sum_{i=1, i\neq j}^{8} \exp(s(z_i, z)/\tau)}, \tag{5.1}$$

where j is the index of the correct answer panel, $s(\cdot)$ is the *cosine similarity* function, and τ is the *temperature* parameter, which we set to 0.1 based on evidence presented in (Chen et al., 2020b).

L_c rewards the model for responding to the correct answer panel with a logits vector z_j that is similar to the prediction z made for the query panel, while penalizing when similar vectors are produced for incorrect answer panels z_i, $i \neq j$, encouraging them to be dissimilar.

Pair Mining Strategy

Finding suitable pairs is a known challenge in contrastive learning (Wang and Isola, 2020), particularly *hard negatives*: pairs whose logits closely resemble the anchor and thus provide the strongest discriminative signal. The RAVEN dataset addresses this naturally by design: incorrect answers are often highly similar to the correct answer, sometimes differing in only one property—precisely the kind of examples that maximize the contrastive signal (Robinson et al., 2021), since soft negatives that already differ substantially from the anchor offer little to discriminate.

The positive examples (correct answer classifications) are not identical to the query predictions, since the model is not initially trained for answer classification. This mismatch is analogous to the role of augmentation in standard contrastive frameworks (Chen et al., 2020b): rather than collapsing onto trivially perfect alignment, the model must tolerate minor representational variation while still identifying the correct answer, providing a richer and more stable training signal.

Training Procedure

CCM fine-tunes a pretrained ACT model on the choice-making task using the contrastive objective defined in Eq. (5.1). During training, the model receives complete RPM tasks including both context panels and all eight answer candidates, with L_c (Eq. 5.1) computed over the resulting predictions z and classifications z_i. When querying a trained model, the answer panel maximizing cosine similarity $s(z_i, z)$ is selected as the model's decision.

CCM employs the same training hyperparameters as LCM (Sec. 5.2.1): Adam optimizer with learning rate 3.5×10^{-5}, batch size 32, and up to 200 epochs with early stopping. Complete training details are provided in Appendix A.1.

5.3.2 Comparative Evaluation

Table 5.13 presents results for both LCM and CCM, each evaluated under two conditions: without augmentation, using the ACT backbone from Table 5.2 (Task tokenizer, Combined masking mode); and with augmentation, using the ACT backbone trained with augmented data from Sec. 4.7. The no-augmentation LCM results are carried over from Table 5.7; all remaining entries are new. Results reveal distinctly different patterns across the three benchmark datasets.

RAVEN and RAVEN-FAIR. On RAVEN and RAVEN-FAIR, LCM and CCM achieve remarkably similar performance results. For RAVEN without augmentation, CCM shows a visible advantage (98.17% vs. 97.44%, a gain of 0.73 percentage points). With augmentation, the difference becomes negligible (98.27% vs. 98.26%). RAVEN-FAIR exhibits the same pattern: without augmentation CCM slightly outperforms LCM (99.27% vs. 99.10%), while with augmentation both reach near-identical ceiling performance (99.30% vs. 99.34%). Overall, determining a clear winner between LCM and CCM on these

Table 5.13: Test-set accuracy of models trained with contrastive loss (CCM) and augmentations compared to the best baseline LCM model.

Method	Augmentation	RAVEN	I-RAVEN	RAVEN-FAIR
LCM	No	97.44	97.14	99.10
	Yes	98.26	96.42	99.34
CCM	No	98.17	95.21	99.27
	Yes	98.27	95.11	99.30

datasets proves difficult—they perform roughly on par, with CCM showing a slight edge without augmentation, despite introducing no additional parameters beyond the pretrained ACT weights.

I-RAVEN Divergent Behavior. I-RAVEN presents markedly different results. CCM performs worse than LCM: 95.21% versus 97.14% without augmentation and 95.11% versus 96.42% with augmentation. In contrast to RAVEN and RAVEN-FAIR, augmentation fails to improve performance for either method and slightly degrades both (LCM: 97.14%→96.42%; CCM: 95.21%→95.11%).

Potential Explanations. At first glance, a natural explanation might appeal to RAVEN's answer set bias (Sec. 3.2.2): incorrect answers differing from the correct one by only a single property resemble hard negatives that strengthen contrastive learning, while I-RAVEN's more diverse distractors provide a softer signal—seemingly disadvantaging CCM (Sec. 5.3.1). This reasoning does not hold, however. All models are trained exclusively on RAVEN regardless of the benchmark they are later evaluated on, so the composition of I-RAVEN's answer sets has no bearing on what CCM learns during training.

A more plausible explanation is that the gap reflects subtle distributional regularities of RAVEN beyond the biases formally cataloged in Sec. 3.2. While the architecture is designed to be resistant to the documented biases, every dataset contains lower-level statistical patterns that shape learned representations. Even with access to answer sets restricted, the model may overfit slightly to these RAVEN-specific regularities—and when those regularities are absent in I-RAVEN, a modest performance gap emerges. That LCM outperforms CCM under this distributional shift may also reflect a structural expressiveness asymmetry: CCM operates entirely within the logit space inherited from ACT pretraining, with no learned parameters to adapt its comparison strategy, whereas LCM's comparator network can compensate for such regularities through its learned similarity function.

5.3.3 Ablation: Representation Choice

ACT's final logits—the pre-softmax outputs of its property prediction head—serve as the representation space for CCM's contrastive objective, but ACT produces several intermediate representations that could in principle serve this role. The question is whether an earlier or differently structured internal representation captures features more suitable for metric comparison than the property-prediction logits themselves.

Projection Heads in Contrastive Learning A common technique across deep learning involves projecting outputs to alternative, typically lower-dimensional spaces, serving multiple purposes: reducing computational costs, preventing overfitting, preventing embedding collapse, and forcing the model to learn more compact representations. In contrastive learning, such *projection heads* have become standard practice (Chen et al., 2020b; He et al., 2020), where they create a separate comparison space that preserves the richer representations in earlier layers while facilitating contrastive objectives. Additionally, distance comparisons between vectors sometimes prove more reliable in lower-dimensional spaces where spurious correlations are reduced. Given these typical benefits of projection, we investigated whether pro-

jecting ACT's logits to a lower-dimensional space through a newly initialized linear layer might improve contrastive comparison quality.

Empirical Evaluation. Table 5.14 compares three alternatives against the baseline logits, all evaluated on RAVEN (Task tokenizer, Combined masking, without augmentation): (i) output of the last hidden layer of the predictor, (ii) raw transformer output without the predictor, (iii) linear projection of predictor's output to 16 dimensions, and (iv) the baseline logits. Contrary to the expectation stated earlier, logits directly yield substantially superior performance (98.17%). Raw transformer output tokens (ii) achieved modest performance (91.91%), indicating that raw attention-based representations lack sufficient semantic structure for contrastive comparison. Hidden layer activations (i) performed better (95.58%), showing property-specific processing improves contrastive learning. Projected logits (iii) performed surprisingly poorly (84.46%), falling below even the transformer output baseline, suggesting that the additional transformation introduces information loss that outweighs any benefits. These results confirm that the *property prediction* space already provides strong representation for contrastive comparison.

Table 5.14: Accuracy of ablated CCM models using different input representations.

Representation fed into CCM	Accuracy (%)
(i) Output of the last hidden layer of the predictor	95.58
(ii) Raw transformer output (no predictor at all)	91.91
(iii) Linear projection of predictor's output to 16 dimensions	84.46
(iv) Logits from predictor's output (baseline, cf. Table 5.13)	98.17

5.4 Bias Resistance Analysis

All three choice makers are structurally prevented from exploiting answer set biases (Sec. 3.2.2), providing architectural grounding for **H2** (Sec. 1.2). DCM achieves this by design; LCM and CCM preserve the same guarantee through strict architectural constraints on their learning phases.

DCM is structurally immune because there is no *trainable* choice-making phase. All learning is confined to *property prediction* in ACT: the index of the correct answer is never used as a target for updating any choice-making parameters, and no gradient flows from the choice outcome into a dedicated decision network. DCM applies a fixed, manually designed similarity measure between the predicted query properties and the classified properties of each answer panel. Any residual sensitivity to dataset biases can only arise indirectly through ACT's property prediction training, which is defined solely on panels and their properties, not on answer sets.

LCM and CCM introduce learning in the choice phase, but both preserve one crucial constraint: *no trainable component ever processes all answer panels jointly.* In both methods, each candidate is scored independently—LCM by passing a single (z, z_i) pair through the dense network, CCM by computing a single cosine similarity $s(z, z_i)$—and aggregation across all eight candidates occurs only analytically, within the cross-entropy or contrastive softmax of the loss function. Because this aggregation is non-learnable, neither method can discover or exploit global answer-set patterns such as positional biases or feature-frequency heuristics; decisions must be grounded solely in how well each individual candidate matches the query representation.

Cross-Benchmark Validation. The structural arguments presented in this chapter receive empirical confirmation through cross-benchmark evaluation. All choice makers (including the trainable LCM and CCM) were trained exclusively on RAVEN and evaluated on RAVEN, I-RAVEN, and RAVEN-FAIR without retraining. Unlike property prediction, where datasets share identical context panels, choice-

making operates on answer sets that differ between datasets. If the models exploited RAVEN-specific answer set patterns, they would fail when these patterns change.

Table 5.13 shows consistent performance across benchmarks for LCM and CCM. DCM results are reported in Table 5.2; even DCM transfers successfully. These results confirm that the architectural constraints preventing answer-set-wide computation remain operative across different answer distributions, validating **H2** (Sec. 1.2) that property prediction mitigates dataset biases.

5.5 Comparison with State-of-the-Art RPM Solvers

This section situates our results within the broader landscape of RPM solvers by comparing ACT-based choice-making modules against published methods on the RAVEN and I-RAVEN benchmarks. We consider both general-purpose deep learning approaches and neuro-symbolic methods that incorporate RAVEN-specific inductive biases, to assess where the framework stands and what architectural trade-offs drive the differences.

Table 5.15: Comparison of RPM solver accuracy on the RAVEN benchmark. Results reproduced from (Małkiński and Mańdziuk, 2025); entries postdating the survey are taken from the original publications.

Method	Acc. (%)
Deep Learning	
Rel-Base (Spratley et al., 2020)	91.7
DCNet (Zhuo and Kankanhalli, 2021)	93.6
CoPINet + ACL (Kim et al., 2020)	93.7
Rel-AIR (Spratley et al., 2020)	94.1
Neuro-Symbolic (RAVEN-specific)	
PrAE (Zhang et al., 2021a)	65.0
NVSA (end-to-end) (Hersche et al., 2023b)	87.7
Rel-SAR (Sun et al., 2025)	96.5
NVSA (attr. labels) (Hersche et al., 2023b)	98.5
ACT+DCM	**97.0**
ACT+LCM	**98.26**
ACT+CCM	**98.27**
Human (Zhang et al., 2019a)	84.4

Our method achieves 98.27% on RAVEN and 97.14% on I-RAVEN, placing it among the strongest general-purpose approaches on both benchmarks. These results substantially exceed human-level performance (84.4% (Zhang et al., 2019a)) and approach the practical ceiling of what the benchmark permits.

Performance on RAVEN Benchmark. Table 5.15 presents selected RPM solvers alongside our choice-making approaches. DCM reaches 97.0%, surpassing all listed deep learning baselines, including Rel-AIR (94.1%), DCNet (93.6%), and CoPINet + ACL (93.7%). LCM and CCM push further to 98.26% and 98.27%, exceeding Rel-SAR (96.5%) and falling within 0.23 percentage points of NVSA with attribute labels (98.5%), despite NVSA relying on RPM-specific symbolic operations and auxiliary attribute supervision.

Performance on I-RAVEN Benchmark. Table 5.16 presents the results for I-RAVEN (Hu et al., 2021). LCM achieves 97.14%, above SCL MLCL+DA (96.8%) and PredRNet (96.5%). All models were trained exclusively on RAVEN, making this cross-dataset transfer particularly informative. DCM (95.4%) and CCM (95.21%) score 1.6–3.1 percentage points lower on I-RAVEN than on RAVEN, reflecting increased difficulty when answer panels lack exploitable patterns, but remain competitive with methods such as SCL (95.0%) and PoNG (95.9%). LCM maintains the highest performance among our approaches,

Table 5.16: Comparison of RPM solver accuracy on the I-RAVEN benchmark. Results reproduced from (Małkiński and Mańdziuk, 2025); entries postdating the survey are taken from the original publications.

Method	Acc. (%)
Deep Learning	
SRAN (Hu et al., 2021)	60.8
MRNet (Benny et al., 2021)	86.8
SCL (Wu et al., 2020)	95.0
PoNG (Małkiński and Mańdziuk, 2025b)	95.9
PredRNet (Yang et al., 2023)	96.5
SCL MLCL+DA (Małkiński and Mańdziuk, 2022)	96.8
Neuro-Symbolic (RAVEN-specific)	
PrAE (Zhang et al., 2021a)	77.0
Learn-VRF (Hersche et al., 2023a)	79.5
NVSA (end-to-end) (Hersche et al., 2023b)	88.1
ARLC (Camposampiero et al., 2024)	92.4
Rel-SAR (Sun et al., 2025)	98.0
NVSA (attr. labels) (Hersche et al., 2023b)	99.0
ACT+DCM	**95.4**
ACT+LCM	**97.14**
ACT+CCM	**95.21**

likely because joint fine-tuning refines ACT's representations while its learned comparator compensates for any remaining deficiencies when transferring across datasets.

Comparison with Neuro-Symbolic Approaches. Neuro-symbolic methods achieve competitive and, in some cases, higher absolute performance by combining neural perception with RAVEN-specific symbolic reasoning. The strongest results come from NVSA (99.0% on I-RAVEN with attribute labels) and Rel-SAR (98.0% on I-RAVEN with rule label supervision). As detailed in Sec. 2.4, these methods encode RAVEN's structure directly into their design: VSA representations target RAVEN's specific attribute domains, relation functions assume RAVEN's rule types, and training often requires auxiliary supervision beyond answer indices. ACT employs property labels for its prediction stage, but uses them to train a general-purpose transformer rather than to populate RAVEN-specific symbolic operations. The result is an architecture whose components are not committed to RAVEN's particular combinatorial structure, reaching 98.27% on RAVEN and 97.14% on I-RAVEN without task-specific engineering. The extent to which this architectural generality transfers to datasets with different structure is examined in Chapter 6.

5.6 Conclusion

This chapter completed the staged self-supervision framework by introducing three choice-making modules that transform ACT's property predictions into answer selections. DCM established that fixed cross-entropy comparison achieves 97.0% accuracy, but classification analysis exposed a central limitation: systematic misclassification of incorrect answer panels undermines the fixed similarity metric, particularly for the Row tokenizer. LCM resolved this through learned similarity functions and joint fine-tuning, reaching 98.26% on RAVEN and 97.14% on I-RAVEN. CCM demonstrated that contrastive learning provides an alternative path to comparable performance (98.27% on RAVEN) without requiring additional architectural components. These results provide evidence for the thesis's principal claims (Sec. 1.2). Staged decomposition into property prediction and choice making (**H1**) enables the specialization that drives performance: each component focuses on a distinct objective, and joint fine-tuning in LCM refines representations in ways unavailable to end-to-end architectures, itself evidence that self-supervised pretraining produced representations worth refining (**H3**). The consistent transfer from RAVEN to I-RAVEN

and RAVEN-FAIR without retraining provides direct evidence for **H2**: training ACT without answer set exposure, and constraining choice-making modules to process candidates independently, prevents the exploitation of dataset-specific biases.

The following Chapter 6 examines abstract reasoning abilities beyond the RAVEN benchmark and ACT framework—testing generalization of RAVEN-trained models on different abstract reasoning datasets, analyzing the limitations of the current framework and their potential remedies, and exploring how large language models engage with RAVEN-style tasks as a diagnostic lens for future directions.

Chapter 6

Additional Investigations and Future Work

This chapter extends the thesis beyond the main **RAVEN** (Sec. 3.1.1) evaluation in two complementary ways. First, it presents additional experiments and analyses that probe the boundaries of the current framework, identifying where it generalizes and where it breaks down. These findings point toward concrete future directions: highlighting the areas that seem most promising, most likely to yield improvements in generality and performance, and most likely to provide insight into what makes abstract visual reasoning hard. Second, the chapter discusses those directions directly, grounding each proposal in the evidence the analyses provide rather than treating it as open-ended speculation. The chapter therefore combines experiments and future work intentionally: the analyses motivate the directions, and the directions give the analyses their purpose.

The experimental part begins with a cross-dataset evaluation on **PGM** (Barrett et al., 2018), which reveals both the extent of the framework's transferable knowledge and the representational boundaries imposed by its manually defined property vocabulary. This is followed by an examination of how the framework could be adapted to structurally related reasoning tasks beyond **RAVEN**. The forward-looking part then builds directly on these findings, proposing directions—automated symbolic vocabulary discovery, differentiable rendering, and enhanced training methodologies—that address the limitations the analyses expose and open what appear to be the most interesting avenues for future investigation. The chapter closes with a parallel analysis for large language models, extending the discussion of their abstract reasoning abilities introduced in Chapter 2: an empirical evaluation of structured prompting strategies, followed by the future directions it suggests.

6.1 Procedurally Generated Matrices (PGM)

The preceding chapters demonstrated that our method, trained exclusively on **RAVEN**, achieves consistent results across all three benchmarks: 98.27% on RAVEN, 97.14% on **I-RAVEN**, and 99.30% on **RAVEN-FAIR** (Sec. 5.5). While I-RAVEN and RAVEN-FAIR introduce out-of-distribution answer

panels, all three datasets share the same geometric figures, symbolic property definitions, and arrangement structures, so these results validate transfer within a common visual vocabulary rather than across distinct visual domains. Whether the learned representations capture geometric and relational concepts that genuinely transfer beyond RAVEN's visual vocabulary, or merely encode RAVEN-specific patterns, requires evaluation on a dataset with different visual elements.

The **Procedurally Generated Matrices (PGM)** dataset (Sec. 3.3.2) offers a natural test case for examining generalization beyond RAVEN. Both datasets feature 3×3 matrices of rule-governed panels where participants select the correct completion from eight alternatives. The panels comprise figure sets with distinctive properties, sharing many comparable geometric attributes. However, PGM presents several key differences: its standard arrangement corresponds to RAVEN's most challenging configuration (the full 3×3 matrix—*distribute-nine*), it incorporates a background layer component absent from RAVEN's design, and it includes visual elements not present in RAVEN's vocabulary, such as star shapes and overlapping figures. Despite RAVEN's greater structural diversity across seven distinct spatial arrangements, PGM poses a more demanding computational problem due to the absence of simpler spatial arrangements and its background layer complexity. This high-level structural similarity creates an opportunity for examining cross-dataset generalization when RAVEN-trained models are applied to PGM tasks. Comprehensive dataset comparisons are available in (Zhang et al., 2019a) and (Małkiński and Mańdziuk, 2025).

We evaluated the **DCM** (Sec. 5.1) and **LCM** (Sec. 5.2) models, using the PGM test set to analyze performance patterns and output characteristics through visualization of reconstruction outcomes. Since the models predict symbolic properties defined for RAVEN, PGM reconstructions appear as RAVEN-style panels derived from PGM input data (the context panels, answer options, and ground truth are visualized using the original PGM dataset, while the model's predictions and reconstructions are generated using RAVEN's visual representation system). This experimental approach yields insights into model behavior when encountering out-of-distribution data with distinct concepts, rules, and entities.

6.1.1 Reconstruction Patterns and Cross-Dataset Transfer

Figures 6.1 and 6.2 present representative examples from the PGM test-set evaluation of DCM Row+Combined (Sec. 5.1) and LCM Row+Combined (Sec. 5.2), both trained exclusively on RAVEN. To distinguish authentic cross-dataset transfer from RAVEN-specific encoding, we conduct a qualitative analysis through visualization of reconstruction outcomes, since both models achieve near-random accuracy reflecting the domain shift between the RAVEN and PGM datasets; quantitative accuracy alone provides limited insight into the nature of the learned representations.

The reconstructed panels demonstrate notable resemblance to input PGM panels, suggesting meaningful generalization through preserved thematic consistency and property values. Similar figures are generated, with colors and shapes shared between datasets reconstructed properly in many cases. Reconstructed figures maintain consistent color, shape, and size attributes while partially preserving original spatial arrangements. This partial transfer illustrates a general property of deep learning representations: even concepts that exist in both datasets are never fully decoupled from their training context. Their representations carry residual coupling to the specific visual characteristics, proportions, and co-occurrence patterns of the data they were learned from, so transfer is consistent where those characteristics overlap and degrades where they diverge.

However, reconstruction quality remains suboptimal due to dataset differences. PGM contains entities absent from RAVEN, including *star shapes* and a background layer with lines, squares, and circles subject to rules but unrecognized by the model.

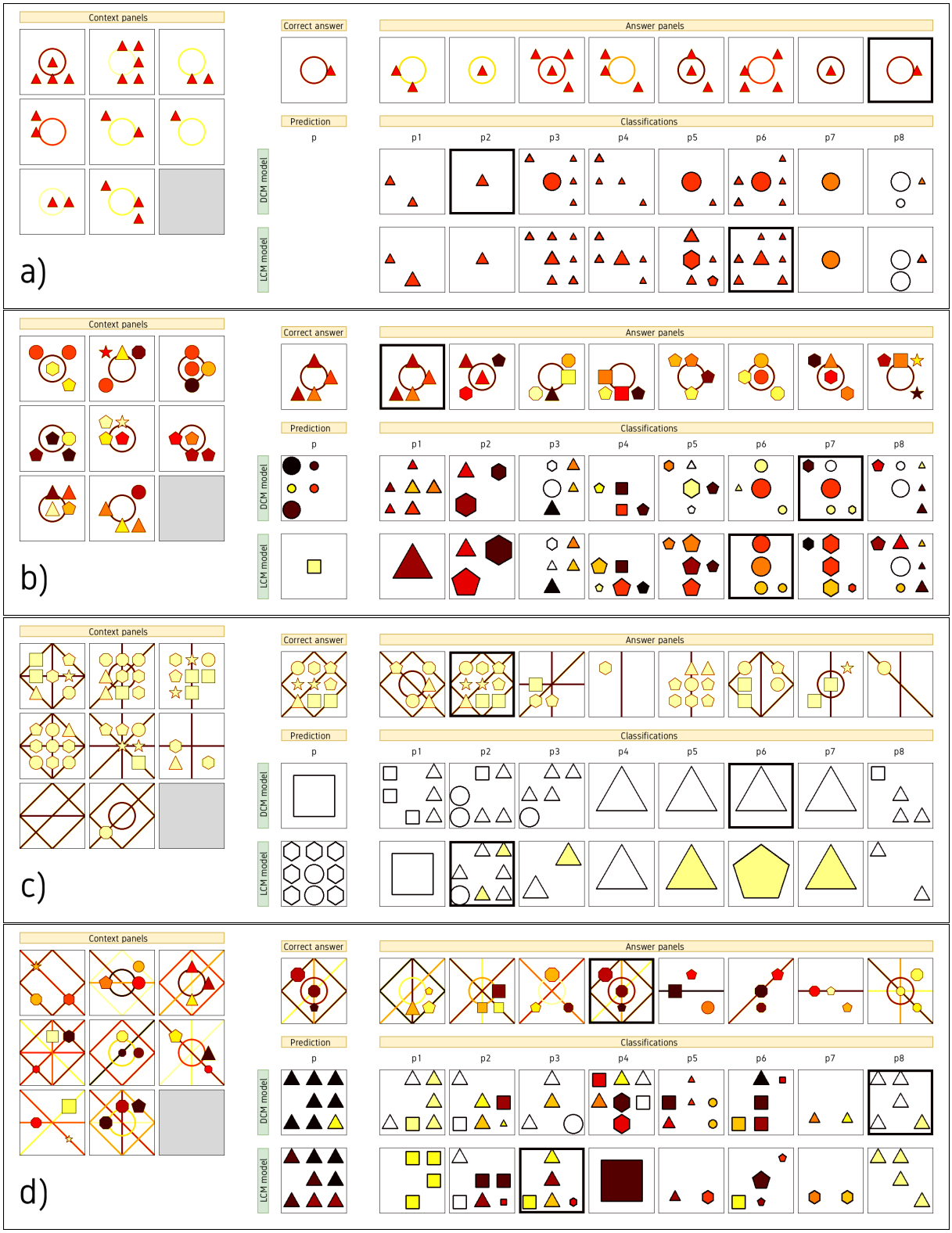


FIGURE 6.1: Generalization analysis of **DCM** trained on RAVEN and evaluated on PGM dataset (Part 1). **Left column:** PGM input task with context panels and answer options. **Middle column:** correct answer and RAVEN-style reconstruction of the masked panel from predicted properties. **Right column:** RAVEN-style reconstructions of all eight answer panels with the selected answer marked by thicker border. Example (b) demonstrates background circle reconstruction where the model integrates visual information across layers. Example (c) shows center-single arrangement mappings where background lines are treated as one large figure in RAVEN's representational framework. Example (d) exhibits similar behavior. Additional examples are provided in Fig. 6.2 and Figs. A.5–A.6.

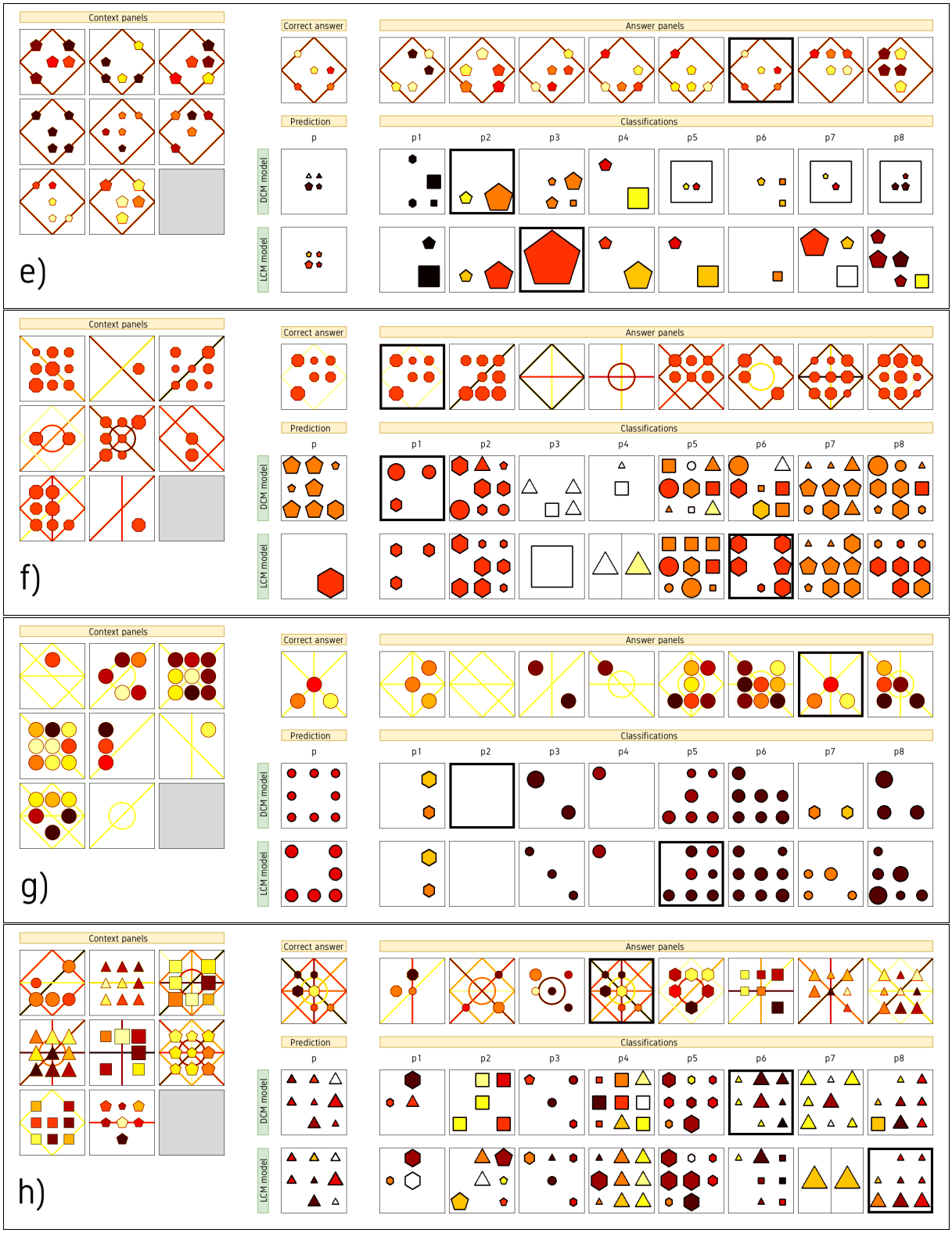


FIGURE 6.2: Generalization analysis of **DCM** trained on RAVEN and evaluated on PGM dataset (Part 2). Example (e) continues demonstrating center-single mapping behavior for background structures. Example (f) reveals geometric extrapolation where the background lines forming a square are successfully mapped to a RAVEN square representation, while neighboring panel p4 shows alternative left-center-single-right-center-single layout selection based on conceptual alignment. Example (g) illustrates systematic omission of unrecognized background elements, where lines-only panel p2 is predicted as empty. Example (h) shows application of Distribute Three rule across datasets, where the model aligns with initial visual impression despite logical inconsistencies.

Since background components are absent from RAVEN training data, the model frequently treats them as noise and systematically omits them during reconstruction. Panel p2 in example (g) illustrates this limitation: only lines are visible, yet the model predicts an empty panel. Lines are present in every panel of the sequence, yet the model consistently omits them from all predictions. PGM's background circles appear as unfilled, outlined shapes of varying sizes that overlap perceptually with other figures, unlike their RAVEN counterparts.

Not all background elements are excluded. When entities exist in RAVEN with similar features and comparable visual characteristics, reconstruction attempts occur. In example (b), panels p6 and p7 contain foreground figures placed inside background circles; the model reconstructs these regions as circles filled with overlapping foreground colors, demonstrating its ability to integrate visual information across layers when familiar shapes are detected. Background circles without any foreground figure—panels p3 and p8 in example (b)—are reconstructed as solid foreground circles rendered in white. A similar pattern appears in example (a): panels p3, p5, and p7 exhibit the foreground-inside-circle reconstruction, while panel p8, containing a background circle without a foreground figure, is reconstructed as a white foreground circle.

In example (c), where context panels contain substantial background layers, the model maps background lines into a *center-single* arrangement, treating them as one large figure in RAVEN's representational framework. These single large figures sometimes adopt the colors of multiple background elements that overlap within the reconstructed region.

Generally, the less cluttered the foreground, the more frequently background lines are detected as arrangements with larger figures. Example (f) illustrates this mapping behavior: when panels contain multiple foreground objects, even those with background lines are still detected as *distribute-nine* arrangements. However, panels p3 and p4, which contain no foreground figures, are detected by the model as different arrangements.

The lines forming a square in panel p3 of example (f) are successfully mapped to a RAVEN square in a *center-single* arrangement representation, demonstrating capacity to recognize geometric relationships with some extrapolation, as such large squares do not exist in RAVEN. RAVEN treats rotational angles as noise attributes, training the model to focus on essential shape properties while disregarding angular variations. Consequently, lines forming an angled square are processed simply as a square entity.

The neighboring panel p4 contains a background circle that is not reconstructed as *center-single*. The circle's smaller dimensions prompt the model to select an alternative mapping: the *left-center-single-right-center-single* layout, driven by vertical line elements matching those in the input PGM image. This panel-to-panel variation reveals competition between diverse visual cues, where the strongest representational match determines reconstruction. The decision process evaluates whether multiple objects resemble RAVEN's *distribute-nine* configuration, or whether background line patterns should dominate the representational signal. This behavior is consistently observed across multiple examples, including examples (d) and (e).

Since RAVEN lacks star shapes, the model consistently reconstructs them as triangles—the most similar available shape. This pattern appears in example (b), with additional instances shown in examples (l), (m), and (n) in Supplementary Materials (Sec. A.2.3), demonstrating the model's tendency to map unfamiliar entities to nearest neighbors within learned representation spaces.

6.1.2 Partial Rule Detection and Reasoning Limitations

In example (h), the visual pattern suggests the *Distribute Three* rule based on shape distribution across rows. Squares appear in all three rows, pentagons in the second and third rows, and triangles in the first and second rows. This distribution could suggest the missing panel should contain triangles, as

they appear in the first and second rows but are absent from the third. However, closer examination reveals a critical inconsistency: the first row lacks pentagons entirely, with circles appearing instead. This contradicts the *Distribute Three* rule and eliminates this as a valid solution pattern.

The model aligns with the initial visual impression and applies the perceived rule accordingly. Both prediction and choice-making components generate triangle-filled panels and select answers that reconstruct triangles correspondingly. While a truly capable reasoning agent should revise its initial assessment after recognizing that pentagons are absent from the first row, the model has nonetheless acquired some conceptual understanding of the *Distribute Three* rule and can apply it across datasets, indicating some level of generalization exists even when trained exclusively on RAVEN.

6.1.3 Implications for Learned Representations

Despite achieving only random-level performance on PGM in terms of absolute accuracy, the visualization analysis reveals important insights about learned representations. The model maps unfamiliar PGM entities to nearest RAVEN counterparts: reconstructing stars as triangles, interpreting background line arrangements as geometric shapes, and partially detecting the *Distribute Three* rule. This demonstrates acquisition of shape concepts rather than reliance on superficial visual features alone or answer panel discrimination strategies. This supports **H2** (Sec. 1.2) regarding bias mitigation through property prediction and **H3** (Sec. 1.2) regarding the success of self-supervised learning.

However, the learned representations remain deeply *RAVEN-conditioned*. The model operates through a hierarchical process: first detecting spatial arrangement, then recognizing figure properties within that arrangement context. This architecture-level coupling becomes evident in cross-dataset transfer. The model frequently misclassifies PGM panels' underlying arrangement types, attempting to map them into RAVEN's predefined categories (*center-single*, *distribute-nine*, *left-center-single-right-center-single*), while still reconstructing recognizable figures. When arrangement classification fails, figure reconstruction operates within an incorrect spatial framework, leading to systematic errors despite preserving some geometric relationships. This behavior reveals a middle ground: the model has acquired conceptual understanding of geometric principles that transfer across datasets with moderate success, yet remains constrained by RAVEN-specific biases in the hierarchical arrangement-first prediction structure.

6.2 Adaptation to Structurally Related Reasoning Tasks

While the PGM analysis highlights representational constraints arising from RAVEN-specific property vocabularies, the staged self-supervision framework exhibits architectural characteristics that suggest potential applicability to structurally similar reasoning tasks through minimal modifications. We outline how components—image tokenization, transformer-based pattern recognition, and property prediction—could be adapted to alternative benchmarks, establishing the framework's conceptual flexibility. These proposals remain theoretical, clarifying which adaptation challenges are architectural and which require symbolic vocabulary extensions addressed in Section 6.3.

Visual Analogy Problem (VAP). VAP (Hill et al., 2019) employs a 2×3 grid with four-option answer sets, differing from RAVEN only in reduced spatial dimensions (see Sec. 3.3.5 for full description). Adaptation is straightforward: tokenizers process two rows instead of three (Row), construct a 2×3 matrix (Task), or handle six panels (Panel). Choice-making requires five forward passes instead of RAVEN's nine.

Abstraction and Reasoning Corpus (ARC). ARC (Chollet, 2019) (Sec. 3.3.3) presents a strong alignment with the property prediction paradigm, as tasks are inherently generative: given 2–3 input-output demonstration pairs, the model must construct the output grid for a new input without predefined

answer options. This structure directly parallels **ACT**'s property prediction phase—the model observes context (demonstration pairs) and generates the target (output grid)—making ARC conceptually compatible with the framework's first stage without requiring choice-making adaptation. Input and output grids can be stacked channel-wise and processed analogously to the Row tokenizer, with each demonstration pair treated as a "panel" and the target output masked during training. The transformer processes tokens from visible demonstrations, learning transformation rules implicit in the examples. The predictor reconstructs the masked output grid directly rather than predicting discrete properties. The central adaptation challenge involves variable grid sizes requiring scaling mechanisms such as dynamic tokenization, padding strategies, or resizing modules to handle arbitrary input dimensions. Optionally, choice-making functionality could be incorporated by generating negative examples through augmentation (incorrect symmetries, wrong object counts, misapplied color mappings), reformulating ARC as a selection task similar to RAVEN. However, this extension is not necessary for basic adaptation: ARC's generative structure already aligns with property prediction. The choice-making phase could potentially serve as a fine-tuning mechanism to improve answer quality through contrastive training, but the base framework operates in generation-only mode for ARC tasks.

Bongard-LOGO. Bongard-LOGO problems (Nie et al., 2020) (Sec. 3.3.4) test concept induction through set-based discrimination rather than sequential pattern completion. Each problem provides a positive set of six panels sharing a target feature and a negative set of six panels lacking it, together with two answer candidates: one belonging to the positive set and one to the negative set. The positive set serves as the context matrix, while the correct answer candidate acts as the query panel, the equivalent of the masked ninth panel in RAVEN. The incorrect answer candidate is added to the six negative panels, forming a seven-option wrong-answer pool; combined with the one correct answer, this yields eight candidates total, matching RAVEN's answer set structure (Sec. 3.1.1). During inference, only three forward passes are required: one with the mask alone, and one each for the two answer candidates, which are then compared against the prediction. One conceptual challenge is that we impose spatial structure on a set whose panel order is arbitrary. Two mitigation strategies address this: (i) applying positional encodings only within individual panels rather than across panel positions (straightforward for the Panel Tokenizer, but requiring modified embedding schemes for Task and Row tokenizers), or (ii) randomly shuffling panel order during training combined with random masking across all positions—a more drastic variant of **RAVEN**'s random masking phase (Sec. 4.3.1)—to discourage reliance on spatial positioning.

Odd One Out (O3). The Odd One Out task (Ruiz, 2011; Mańdziuk and Żychowski, 2019) (Sec. 3.3.5) requires the most substantial adaptation among the tasks considered here, as it inverts the ACT objective entirely: rather than constructing a missing panel, the goal is to identify the one panel that does not belong. Like Bongard-LOGO, O3 operates on an unordered set, and the same positional mitigation strategies described above apply.

We adapt the framework in two stages. In the first stage, we train ACT on sets of four conforming panels using random masking, establishing property predictions over homogeneous sets. In the second stage, we train the choice-making module: for each of the four slots in the conforming matrix, we run three passes—mask only, mask replaced with the conforming panel, mask replaced with the anomalous panel—yielding 12 runs per sample. Running all four slots provides a richer training signal than a single slot would, since each slot offers an independent comparison between a conforming and an anomalous candidate; however, a single slot would suffice in principle, as in RAVEN. The loss function (binary cross-entropy for LCM, pairwise contrastive loss for CCM) is computed per slot.

Inference requires a different strategy, because the anomalous panel is unknown: we cannot construct a clean conforming context matrix in advance, as identifying the odd one out is the task itself. We therefore evaluate all five leave-one-out configurations. For each, one panel is held out and the remaining

four form the context matrix. We then run three passes at a single fixed slot; by analogy with RAVEN, we use the last position, replacing it with the mask, with the panel occupying that slot, and with the held-out panel. The configuration in which the held-out panel is the anomalous one produces the largest gap: the conforming replacement yields low reconstruction error while the anomalous held-out panel yields high error. In all other configurations, the context is polluted by the anomalous panel still inside the matrix, and both replacements produce similar errors, collapsing the gap. The panel whose removal produces the largest gap between the two replacement scores is classified as the odd one out. This requires $5 \times 3 = 15$ forward passes at inference. Extending to all four slots would increase this to $5 \times 3 \times 4 = 60$ passes but is unlikely to yield substantial gains, given that positional mitigation training is specifically designed to make any single slot representative.

Implications for Framework Flexibility. These adaptations demonstrate inherent flexibility when task structures align with rule-based reasoning. Tasks differing primarily in spatial layout (VAP), task objective (O3), problem formulation (Bongard), or generation paradigm (ARC) require modifications ranging from trivial (VAP: dimension changes, ARC: padding for variable grid sizes) to moderate (O3/Bongard: restructured task formulation, ARC: optional choice-maker negative sample generation). The primary consideration for new domains is defining an appropriate symbolic property vocabulary, which motivates the automated discovery approach in Section 6.3.

6.3 Future Work: Automated Discovery of Visual Symbolic Representations

The preceding analyses establish both the framework's strengths and its boundaries. The PGM cross-dataset evaluation demonstrates that models trained on RAVEN acquire tangible geometric concepts that transfer partially to novel visual vocabularies—preserving color, shape, and size attributes for entities shared between datasets, and partially detecting rule patterns such as Distribute Three—yet mismatched arrangement inference and out-of-vocabulary entities force approximate symbolic mappings. The adaptation proposals in Section 6.2 demonstrate that while architectural flexibility enables minimal-modification adaptation to structurally similar tasks, extending to domains with unknown or diverse visual vocabularies benefits from automated symbolic representation discovery.

These findings suggest three natural extensions that preserve the framework's core advantages—bias mitigation via **property prediction** (**H2**, Sec. 1.2) and interpretability via explicit intermediate representations—while enabling broader applicability: (i) learning symbolic vocabularies from data rather than manual design, (ii) strengthening decoder compositionality through differentiable drawing architectures, and (iii) improving training robustness through curriculum learning, contrastive objectives, and knowledge distillation. We prioritize the first direction, as automated vocabulary discovery addresses the representational engineering bottleneck while preserving the staged decomposition principle that enables the framework's current successes.

6.3.1 Automated Symbolic Vocabulary Discovery Through Vector Quantization

The ground truth target properties serve as a structured interface between visual perception and abstract reasoning, enabling the transformer to learn logical progressions while resisting biases. Automating the discovery of such properties would extend this advantage to datasets where appropriate symbolic descriptors are unknown *a priori*, or where manually curating them for every new visual domain would be impractical.

We propose adapting the **Vector Quantized Vision Transformer (VQ-ViT)** architecture (Yu et al., 2022; Peng et al., 2022) to automatically learn symbolic representations of RPM panels in place of manually curated property vocabularies. This model consists of three components:

- **Encoder Module**: A Vision Transformer encoder (Dosovitskiy et al., 2020) that processes input panels and maps them to a continuous latent space, capturing hierarchical visual features from low-level geometric primitives to high-level compositional relationships.
- **Vector Quantization Layer**: A discrete bottleneck (van den Oord et al., 2017) that quantizes continuous representations into a finite vocabulary of symbolic tokens, where each token corresponds to a learned visual concept or compositional pattern.
- **Decoder Module**: A decoder that reconstructs the original panel from the quantized representation, ensuring that learned symbolic tokens retain sufficient information for accurate visual reconstruction.

Integration, Efficiency, and Cross-Dataset Generalization

The learned symbolic representations replace the target property vocabulary in the ACT framework. The VQ-ViT is first trained on a comprehensive corpus of RPM panels to learn a general-purpose symbolic vocabulary, which then serves as the target representation space for the ACT transformer—predicting quantized symbolic descriptions of missing panels from logical patterns in context panels. This maintains the interpretability benefits of the original framework, and the discrete representations enable efficient transformer processing while preserving the structural information required for abstract reasoning.

Once trained, the VQ-ViT encoder generates symbolic representations for all panels in a single forward pass; the ACT transformer can then train exclusively on these pre-computed tokens, eliminating repeated visual processing. The representations can be cached and reused across experiments, reducing overhead during hyperparameter optimization. Reattaching the decoder should enable verification of reconstruction quality through direct visual comparison, providing insight into which panel elements are properly reconstructed.

By learning symbolic representations in a data-driven manner, the VQ-ViT should adapt to diverse visual vocabularies across RPM datasets. While visual surface features differ between **RAVEN** and **PGM**, underlying abstract relationships often follow similar structural patterns. We expect that representations learned from a single dataset will carry residual coupling to that dataset's specific visual vocabulary; exposure to a greater diversity of visual reasoning tasks should progressively reduce this coupling, producing symbolic tokens that approximate dataset-agnostic concepts rather than RAVEN-specific encodings. Beyond training diversity, the discrete bottleneck itself provides a complementary mechanism that we believe would further reduce this coupling. Continuous representations offer gradient descent the flexibility to encode both actual conceptual structure and dataset-specific residuals simultaneously, with no architectural pressure separating them. A finite discrete codebook removes this flexibility at the critical representational bottleneck: token slots are a limited resource, and we expect optimization pressure to favor reusable, task-relevant representations over dataset-specific ones. This is analogous to KL regularization (Kullback and Leibler, 1951) in VAEs (Higgins et al., 2017), which encourages dataset-agnostic structure by penalizing deviations from a fixed prior; the discrete bottleneck would achieve this through architectural constraint rather than an auxiliary loss term. Crucially, the straight-through estimator preserves gradient flow through the quantization step, so the entire architecture remains trainable with standard stochastic gradient descent. This is precisely the principle of *implicit symbolic processing* discussed in Chapter 2: a model that remains trainable end-to-end via stochastic gradient descent, yet gains the generalization advantages of discrete structure at a strategically chosen point in the architecture.

6.3.2 Differentiable Drawing Modules and Symbolic Rendering

The proposed framework presents opportunities for incorporating **differentiable drawing modules** as decoder architectures, extending beyond conventional reconstruction approaches. Recent advances in differentiable rendering have introduced neural modules capable of generating geometric primitives through learnable, gradient-compatible operations (Kato et al., 2020; Liu et al., 2020; Liang, 2022), offering a promising approach for RPM panel reconstruction.

Differentiable Symbolic Drawing Systems leverage parametric representations of geometric primitives (circles, squares, lines, and polygons) that can be rendered through differentiable rasterization processes (Petersen et al., 2022). These systems maintain explicit symbolic parameters (center coordinates, radii, edge lengths, orientations) while enabling end-to-end gradient flow from pixel-level reconstruction losses back to high-level geometric parameters. This dual symbolic-neural nature makes them well-suited for abstract reasoning tasks where interpretability and compositional understanding are paramount. This approach creates a natural *symbolic language* for panel description that operates at the level of geometric primitives rather than raw visual features. Concretely, the decoder generates sequences of parameterized commands such as "place a rectangle of width `w` and height `h` at coordinates `(x,y)`," "draw a circle of radius `r` centered at `(cx,cy)`," or "fill a polygon with color `c`."

Replacing the standard raster decoder with a drawing module provides two concrete advantages over the VQ-ViT reconstruction approach. First, the *geometric primitive vocabulary* (circles, rectangles, lines, polygons, and their transformations) operates at a more fundamental level of visual abstraction than either dataset-specific feature representations or learned codebook tokens, enabling cross-dataset transfer where models trained on simple shapes can generalize to more intricate visual elements with similar compositional structure. Second, the explicit drawing instruction sequences provide interpretable traces of the reasoning process—covering which geometric elements are relevant, how they combine to represent relationships, and their spatial ordering—enabling **counterfactual analysis** by systematically perturbing geometric parameters or primitive selections to map causal relationships between visual features and abstract reasoning rules.

6.3.3 Enhanced Training Methodologies

Beyond architectural innovations, several training enhancements could improve robustness and generalization. These include curriculum learning through progressive masking, contrastive objectives that emphasize logical consistency, and knowledge distillation that transfers reasoning capabilities from larger teacher models.

Curriculum Learning Through Progressive Masking

Masking panels during training forces the model to predict missing properties from context alone, but masking need not operate at the granularity of complete panels. Individual tokens within a panel can be masked instead, allowing arbitrary portions of the visual context to be withheld. Since each panel is represented as a sequence of tokens, distributing masked tokens across multiple panels partially occludes several panels at once rather than fully removing any single one, requiring fine-grained relational inference from incomplete local evidence.

This token-level flexibility lends itself to a *curriculum learning* strategy (Bengio et al., 2009). Training begins with a low masked token budget spread across the matrix, providing near-complete context while the model acquires foundational visual concepts. The budget is then gradually increased across varying panel combinations, confronting the model with progressively more incomplete information and demanding increasingly elaborate relational inference to recover the missing structure. This gradual exposure

builds rich internal representations and strengthens the ability to infer logical progressions across the matrix.

Contrastive Learning and Data Augmentation

Modern self-supervised learning has been advanced by **contrastive learning** methodologies that learn representations by distinguishing between positive and negative examples (Chen et al., 2020b). In visual reasoning, contrastive approaches could emphasize logical consistency while penalizing violations of abstract rules, through strategies such as introducing logical inconsistencies in matrix progressions, generating incorrect panel sequences, or applying augmentations (geometric transformations, color space manipulations, compositional variations) that preserve underlying rule structure while varying surface-level visual features. These approaches encourage learning representations invariant to irrelevant visual variations while remaining sensitive to logically significant changes, and the resulting representations are more likely to transfer across datasets where surface appearance differs but abstract rule structure is shared.

Knowledge Distillation Through Teacher-Student Architectures

Knowledge distillation (Hinton et al., 2015) transfers reasoning capabilities from a larger, higher-capacity teacher model to a smaller student model, aiming to preserve performance while reducing computational cost. The ACT teacher model's intermediate representations provide rich supervisory signals that guide the student toward capable visual and relational encoding. More importantly, the probability distribution over answers produced by the teacher's choice-making module constitutes a more informative training signal than a one-hot target label: it encodes the teacher's relative confidence across all candidates, capturing partial structural similarities that binary supervision discards. When the student model has comparable capacity to the teacher, this richer signal has the potential to improve generalization beyond what direct supervision achieves, as the soft targets act as a form of regularization that smooths the learned decision boundary (Kim et al., 2021).

6.4 Large Language Models as Abstract Reasoners

The preceding subsections examined future directions for **ACT** as a trained architecture: extending its symbolic vocabulary through vector quantization, strengthening its decoder through differentiable rendering, and improving its training regime through curriculum learning and knowledge distillation. A complementary and diagnostically distinct question asks how far the *opposite extreme* can go on the same tasks: a general-purpose large language model that receives no task-specific training and is guided only through structured inputs. This connects directly to the limitations identified in Section 2.3.2: prompt-augmented models can generate the *appearance* of reasoning while relying on the same statistical pattern matching underneath. Whether structured prompting genuinely activates new reasoning capacity or merely surfaces pre-existing knowledge depends on one concrete question: do LLMs possess a mechanism for retaining and building upon intermediate results across reasoning steps; in short, a form of **working memory** (Sec. 6.4.1)?

6.4.1 Working Memory in Neural Architectures

Abstract reasoning tasks place particularly high demands on any mechanism for retaining and revisiting intermediate results: solving an RPM requires tracking candidate rules across multiple attributes simultaneously, revising partial hypotheses when later panels introduce contradictions, and narrowing the solution space incrementally rather than in a single inference step. We refer to this capacity as *working*

memory (Landi et al., 2021), temporary, selectively accessible storage that allows a reasoning system to decompose a problem, preserve partial conclusions, backtrack if a direction proves unproductive, and redirect attention without compressing the full reasoning chain into a single forward pass. The simplest attempt at this is the **residual connection** (He et al., 2016), which retains information passively across layers, but uniformly and without selectivity, so the network has no means of deciding what is worth keeping. **Long Short-Term Memory** (LSTM) networks (Hochreiter and Schmidhuber, 1997) and **Gated Recurrent Units** (GRUs) (Chen et al., 2018) addressed this limitation by introducing explicit input, forget, and output gates that allow selective reading from and writing to a persistent cell state: information can be stored at one step and retrieved many steps later without influencing every intermediate computation. More sophisticated architectures extend this further to external differentiable memory modules with explicit addressing (**Memory-Augmented Neural Networks** (MANNs) (Graves et al., 2014) such as Neural Turing Machines and Differentiable Neural Computers) at the cost of components that are difficult to integrate into end-to-end training. Large language models offer a qualitatively different and more flexible realization of the same principle: by externalizing intermediate reasoning steps into the context window (used as a *scratchpad* (Nye et al., 2021)) and relying on the attention mechanism to selectively retrieve the most relevant portions, they achieve working memory without a dedicated memory module, without specialized addressing hardware, and with the full generality of natural language as the storage medium. **Chain-of-thought prompting** and **prompt chaining**, introduced below, are the concrete realizations of this mechanism that this section evaluates.

Chain-of-thought as working memory. Chain-of-thought prompting (Wei et al., 2022b) is the primary mechanism through which this functional working memory is activated: the model produces explicit intermediate reasoning before committing to a final answer, and because generation is autoregressive, each step is appended to the context window and becomes part of the input for all subsequent tokens. Via the transformer's attention mechanism, the model can selectively focus on the most pertinent parts of its own prior reasoning at each step, allowing partial deductions to accumulate and constrain later ones: earlier, simpler conclusions anchoring and narrowing the hypothesis space for the more sophisticated ones that follow. In abstract reasoning this enables a hierarchical strategy: first identifying the simplest attribute rule (color relations are typically more straightforward than more elaborate spatial arrangements), then recording the partial solution before proceeding to the next attribute, so that each confirmed deduction progressively narrows the hypothesis space for all remaining attributes and for the final answer selection.

Prompt chaining as extended working memory. For tasks where reasoning steps exceed what can be reliably managed within a single response, chain-of-thought extends to prompt chaining (Wu et al., 2022): intermediate results are explicitly preserved across separate inference calls, each of which receives not only the original problem context but also the accumulated output of all prior reasoning steps, extending working memory beyond the boundary of a single prompt-response pair. More sophisticated variants could incorporate structured *information compression* between steps, passing a focused summary rather than the full prior context, more closely approximating the selective retention that characterizes working memory.

In-context examples as reasoning scaffolds. The same context window that stores intermediate reasoning steps can accommodate a second form of augmentation: **in-context examples** (Brown et al., 2020; Wei et al., 2022b)—worked solutions provided directly within the input. This shifts from traditional **imitation learning**, which requires specialized datasets and training from scratch, toward leveraging general-purpose models at inference time. Evidence suggests such examples can enhance problem-solving capabilities (Dong et al., 2022), though whether this transfers to abstract rule induction—where examples may anchor the model to familiar solution patterns rather than enabling reasoning about the current instance—remains an empirical question the evaluation below addresses directly (Levy et al., 2022).

Empirical scope. How far this context-window strategy scales remains an open question beyond the scope of this evaluation. Whether linguistic tokenization is the most efficient representational medium for abstract reasoning—or whether more structured multimodal memory systems embedded within the architecture might eventually yield superior performance—is likewise left for future work.

The theoretical grounding developed above leads to several predictions we aim to evaluate. The key prediction of the working memory hypothesis is that any form of intermediate reasoning generation will substantially outperform strategies that rely on declarative knowledge alone: the act of producing explicit reasoning steps, regardless of their structure, should be the primary driver of performance. Beyond this, we expect that additional structural guidance will provide further, if more modest, improvements. A structured protocol gives the model an explicit sequence of steps to follow, reducing the risk of skipping attributes or losing track of partial conclusions. Worked examples provide in-context demonstrations of how to approach the task, offering both a reasoning template and concrete illustrations of the rule vocabulary in use. We therefore expect structured CoT and example-augmented strategies to improve over free-form generation, though we anticipate these gains to be smaller than the step from no intermediate generation to any intermediate generation. Finally, we expect the benefit of intermediate generation to grow with model scale, as larger models should be better positioned to exploit the accumulated reasoning context.

6.4.2 Experimental Setup

This experiment evaluates the ability of large language models to solve **I-RAVEN** (Sec. 3.2.3) tasks, and investigates which prompting strategies improve performance, with a particular focus on testing the working memory hypothesis that intermediate reasoning generation is the leading driver of accuracy in abstract reasoning tasks.

Due to the high computational requirements of evaluating multiple prompting strategies across full model families, we conduct the evaluation on a 700-item subset of **I-RAVEN**, covering all seven spatial arrangement types (100 items per arrangement). We evaluate three model sizes from the Qwen3.5 family (Qwen Team, 2026): 0.8B, 4B, and 27B, running all models in no-think mode—a deliberate choice that disables the model's internal chain-of-thought, so that any intermediate reasoning is produced only when explicitly elicited by the prompting strategy. This allows a clean comparison between conditions where intermediate generation is absent and conditions where it is structurally invited.

Puzzle representation. Each I-RAVEN puzzle is stored as a structured XML file containing the full symbolic description of all panels (object types, sizes, colors, and positions) along with the eight answer candidates. This representation is read directly and serialized into a structured text description. Each panel is described using its attribute values and, for multi-object arrangements, the occupied grid positions (e.g. for top left `TL=large orange square`). Multi-slot layouts use a three-level separator hierarchy: commas separate objects within a panel, semicolons separate panels within a row, and newlines separate rows. Two-layout arrangements (e.g. inner-outer or left-right) are joined by a vertical bar. The eight answer candidates are listed with numeric labels `[0]-[7]`. A complete illustrative prompt is provided in Appendix A.5.2.

Evaluation procedure. For each puzzle, the model receives the prompt for the given strategy and is required to produce either a single digit (strategies S1–S2, and the final call of S7–S8) or a reasoning trace followed by a single digit (strategies S3–S6). The digit identifies the model's chosen answer candidate from the eight options. For chaining strategies (S7–S8), the model receives three sequential prompts: the first two elicit intermediate reasoning about subsets of rows, and the third—augmented with the outputs of the previous calls—requires the final answer digit. Responses are parsed automatically: the last digit

in the range 0–7 appearing in the model's output is taken as the answer. Accuracy is computed as the proportion of correctly answered puzzles over the full evaluation set.

Eight prompting strategies form a controlled ablation. Each strategy builds incrementally on the previous one, either adding a new component or replacing an existing one with a more informative variant, so that each accuracy change can be attributed to a specific manipulation. All strategies share the same underlying symbolic puzzle description and the same full background context block covering the RPM task structure, arrangement types, rule vocabulary, and attribute taxonomy. Differences between strategies therefore reflect reasoning structure and prompt engineering choices, not differences in declarative knowledge.

Two prompting paradigms are represented. **Chain-of-thought (CoT) prompting** (Wei et al., 2022b) elicits intermediate reasoning within a single model response. **Prompt chaining** (Wu et al., 2022) distributes reasoning across multiple sequential calls, where the output of each call is appended to the context of the next. Strategies S3–S6 realize the CoT paradigm; strategies S7–S8 realize prompt chaining.

The goal of this experiment is not to maximize absolute performance on **I-RAVEN**, but to isolate the contribution of specific reasoning components—chain-of-thought generation, prompt chaining, prompt engineering, and in-context examples—by varying them in a controlled manner. The prompt representation, model family, and chaining structure were chosen to make these comparisons clean, not to optimize accuracy. Higher performance would likely be achievable through more extensive prompt engineering, a more expressive puzzle representation, stronger models, or more sophisticated multi-call architectures; the results below should therefore be read as evidence about the relative value of reasoning components, not as a ceiling on what LLM-based approaches can achieve on abstract reasoning tasks.

S1 — Simple baseline, answer-only (zero-shot). Full task context delivered as a flat prompt with a minimal output instruction. No persona, no section structure, no reasoning invitation. This establishes the baseline for what the model can do with correct task framing and no further prompt engineering.

S2 — Prompt-engineered, answer-only (zero-shot). Identical factual content to S1, but delivered with standard prompt engineering techniques (Sahoo et al., 2024): an expert persona, structured section separators, an explicit enumeration of all eight answer candidates to counteract positional preference biases, and a reinforced output instruction. No reasoning is elicited. Comparison with S1 isolates the contribution of prompt engineering over plain delivery of the same information.

S3 — Free-form chain-of-thought (zero-shot). Background context retained; the model is instructed to reason freely in whatever form it finds most natural before stating its answer, without prescribing any specific reasoning sequence. The minimal working memory strategy: it invites intermediate generation without imposing structure. S3 therefore establishes a baseline for what unstructured CoT can achieve.

S4 — Structured chain-of-thought (zero-shot). The model follows a structured six-step protocol: (1) identify the arrangement type; (2) analyze each attribute (type, size, color) row by row; (3) analyze number and position for multi-object arrangements; (4) predict the missing panel's properties; (5) match the prediction against all eight candidates; (6) state the answer. Comparison with S3 isolates the contribution of explicit reasoning structure over unstructured generation.

S5 — Structured CoT with one worked example (one-shot). S4 is augmented with a single complete solved puzzle demonstrating the full reasoning trace before the target puzzle, following the one-shot in-context learning paradigm (Brown et al., 2020).

S6 — Structured CoT with seven worked examples (few-shot). The single worked example of S5 is replaced with seven, one per spatial arrangement type, following the few-shot in-context learning

paradigm (Brown et al., 2020).

S7 — Prompt chaining with one worked example (three calls, one-shot). The puzzle is solved across three sequential model calls: Call 1 analyzes rows 1–2 and records detected rules; Call 2 receives the original puzzle plus Call 1 output and analyzes row 3; Call 3 produces the final answer. One worked example of the full three-call procedure is provided. Note that S7 and S8 depart from the strictly linear progression of S1–S6: rather than extending S6, S7 takes S5 as its base and adds prompt chaining, while S8 takes S6 as its base and adds prompt chaining. The two chaining strategies are therefore parallel variants of S5 and S6 under the chaining paradigm.

S8 — Prompt chaining with seven worked examples (three calls, few-shot). Identical to S7 in structure, augmented with seven worked examples, one per arrangement type, each demonstrating the full three-call reasoning sequence.

The complete prompts for all strategies are lengthy and would occupy disproportionate space in the thesis text, so one illustrative example is included in Appendix A.5.2, and the full set of prompt templates is provided in the accompanying code repository. The strategy descriptions above closely reflect their content: each prompt adds precisely the component described and nothing more, so the general methodology can be reconstructed from the descriptions alone. Table 6.1 summarizes the key properties of each strategy.

Table 6.1: Summary of prompting strategies and key properties. All strategies share the same background context block. *WM scope* indicates the extent of working memory externalization: *none* (implicit only); *single* (within one response); *chained* (across multiple calls).

ID	Label	Paradigm	Examples	Calls	WM scope
S1	Simple baseline	–	0	1	none
S2	Prompt-eng., ans-only	–	0	1	none
S3	Free-form CoT	CoT	0	1	single
S4	Structured CoT	CoT	0	1	single
S5	Structured CoT + 1 ex.	CoT	1	1	single
S6	Structured CoT + 7 ex.	CoT	7	1	single
S7	Prompt chaining + 1 ex.	Chaining	1	3	chained
S8	Prompt chaining + 7 ex.	Chaining	7	3	chained

6.4.3 Results

Table 6.2 presents overall accuracy for all strategies across three model sizes from the Qwen3.5 family: 0.8B, 4B, and 27B. Per-arrangement breakdowns are provided in Table A.18 in the appendix.

Table 6.2: Overall accuracy (%) across all models and all prompting strategies on the 700-item I-RAVEN evaluation set. Best result in each model column is shown in **bold**.

ID	Strategy	0.8B	4B	27B
S1	Simple baseline	13.9	15.0	24.9
S2	Prompt-eng. / ans-only	13.7	14.1	23.4
S3	Free-form CoT	12.7	63.4	82.4
S4	Structured CoT	11.9	**66.1**	**85.0**
S5	CoT + 1 example	13.6	60.7	83.7
S6	CoT + 7 examples	12.6	58.4	81.1
S7	Chain + 1 example	12.1	55.7	82.7
S8	Chain + 7 examples	**15.1**	60.3	81.4

Results are strongly scale-dependent: at 0.8B, all strategies remain near the 12.5% random baseline (11.9–15.1%). At 4B, no-reasoning strategies (S1: 15.0%, S2: 14.1%) stay near random while all

reasoning strategies rise to 55–66%, with S4 the strongest at 66.1%. At 27B, the same split amplifies: no-reasoning baselines reach only 23–25%, while reasoning strategies cluster between 81.1% and 85.0%.

Scale threshold and the role of model capacity. The 0.8B model cannot make productive use of any external reasoning trace: its performance is indistinguishable from random across all eight strategies. This indicates that, at least under our symbolic text-based evaluation protocol, a minimum model capacity is required before the working memory mechanism becomes viable at all. The 4B model crosses this threshold and is therefore the most informative regime for analyzing the relative contribution of structure, examples, and chaining: it is strong enough for chain-of-thought to work, but not so capable that it renders the task trivial. At 27B, all intermediate-generation strategies (S3–S8) become similarly successful and the spread among them narrows to approximately 4 pp (S4: 85.0% vs. S6: 81.1%), suggesting that larger models become increasingly robust to suboptimal prompting choices within this family of strategies.

The working memory gap: S1 and S2 versus S3–S8. The most consequential finding is the near-identical performance of S1 (15.0%) and S2 (14.1%) at 4B, followed by the 49.3 pp jump to S3 (63.4%). Both S1 and S2 deliver the same full background context; S2 additionally applies standard prompt engineering techniques including an expert persona, structured section separators, and an explicit anti-bias enumeration of all eight candidates. Yet neither strategy, lacking any mechanism for step-by-step intermediate generation, produces more than a marginal improvement over random. S3 adds no new declarative content over S2 and no additional prompt engineering. It solely instructs the model to generate intermediate reasoning before answering. This instruction alone more than quadruples accuracy, constituting the principal empirical support for the working memory hypothesis.

S3 to S4: structure is arrangement-dependent, not universally beneficial. Adding the structured six-step protocol (S4: 66.1%) yields a 2.7 pp overall gain over free-form CoT (S3: 63.4%), but the arrangement-level breakdown (Table A.18) reveals a more nuanced picture, though with only 100 items per arrangement, differences should be interpreted cautiously. S4 leads on enumeration-intensive arrangements: distribute-nine (D9: +20 pp), distribute-four (D4: +9 pp), and center-single (CS: +7 pp), where the protocol's attribute-by-attribute discipline forces the model to track objects systematically. On the in-distribute-four-out-center-single arrangement (IDO), where objects are distributed across two nested spatial layers rather than a single uniform grid, S3 outperforms S4 by 14 pp: a rigid six-step sequence over-constrains the model on this arrangement, where flexible holistic analysis of the nested spatial relationship works better than mechanical attribute enumeration. Arrangement difficulty is therefore not a fixed property of the puzzle type but a joint property of the puzzle and the reasoning protocol.

S4 to S5–S6: one-shot and few-shot examples are monotonically harmful at this scale. Adding one worked example (S5: 60.7%) produces a −5.4 pp decline relative to S4; adding seven (S6: 58.4%) produces a further −2.3 pp decline. This monotonic degradation is consistent across all seven arrangement types: no arrangement benefits from examples at 4B. Even the unstructured free-form S3 (63.4%) outperforms both S5 and S6, confirming that the harm is not offset by the structured protocol. One possible explanation is a *retrieval-over-reasoning* failure mode: given that the structured CoT protocol may already provide the model with everything it needs to solve puzzles under this evaluation protocol, the addition of worked examples introduces a complete solution template that the model gravitates toward, suppressing the generative bottom-up reasoning that makes zero-shot strategies successful. This hypothesis is specific to the current evaluation setting and dataset; with more demanding puzzles or a different prompt representation, in-context examples might instead provide substantive reasoning support (Brown et al., 2020). At 27B, the same direction holds but weakens (S5: 83.7%, S6: 81.1% vs. S4: 85.0%), consistent with this effect being scale-dependent.

S7: prompt chaining underperforms single-response CoT at this scale. S7 (55.7%) is the weakest reasoning strategy at 4B, underperforming all single-response CoT variants including free-form S3. The per-arrangement breakdown (Table A.18) shows the failure is arrangement-specific: distribute-four (D4) is nearly unaffected (−1 pp vs. S4), while distribute-nine (D9) collapses by −22 pp and in-distribute-four-out-center-single (IDO) by −16 pp. The IDO score for S7 (36%) is the single lowest score of any reasoning strategy on any arrangement at 4B, in stark contrast to S3's IDO of 66%—a sharp illustration of how compounding errors across frozen call boundaries can be catastrophic for arrangements requiring flexible, holistic analysis. A plausible interpretation, consistent with the retrieval-over-reasoning account above, is that the task complexity under this evaluation protocol does not require the depth of decomposition that chaining imposes; the three-call boundary prevents retrospective correction and amplifies early errors. Importantly, this need not reflect an inherent limitation of prompt chaining as a paradigm: a different decomposition strategy, intermediate summarisation between calls, or adaptive call boundaries may well recover or exceed the performance of single-response CoT. Notably, S8 at 4B (60.3%) partially recovers relative to S7 (55.7%), suggesting that broader example coverage provides some compensating signal in the chaining setting—a pattern that runs counter to the CoT trend and may warrant further investigation.

6.4.4 Conclusions and Implications

Our core prediction—that intermediate reasoning generation would be the primary driver of performance—is confirmed, and more strongly than anticipated. The gap between strategies that produce no intermediate reasoning (S1, S2) and any strategy that does (S3–S8) is between 49 and 52 pp at 4B, and between 58 and 62 pp at 27B. This pattern holds across all intermediate-generation strategies, across two capable model sizes, and across all seven arrangement types. The benefit grows consistently with scale: at 27B the spread among intermediate-generation strategies narrows to approximately 4 pp, suggesting larger models become increasingly robust to suboptimal prompting choices, while the 0.8B model shows no substantive gain from any strategy, confirming a minimum capacity threshold below which the working memory mechanism does not activate. While this evaluation covers one substantially reduced dataset and one experimental setup, the breadth and magnitude of the effect suggest that chain-of-thought reasoning genuinely helps abstract visual reasoning, and possibly transfers beyond this specific setting.

The secondary prediction—that structural and contextual extensions would improve over unstructured CoT—is only partially supported. Structured CoT (S4) improves slightly over free-form CoT (S3), but one-shot and few-shot examples produce a monotonic decline at both 4B and 27B, and prompt chaining underperforms single-response CoT at both scales. Whether this reflects a limitation or an artifact of the current puzzle representation and difficulty level remains open.

Although the LLM-based evaluation presented here differs substantially from the **ACT** experiments, the underlying principle is the same: making intermediate reasoning explicit—whether through generated text in a large language model or through structured property prediction in a trained architecture—is more capable than compressing the full reasoning chain into a single implicit computation. This demonstrates how architectural simplicity combined with scale can match or surpass more theoretically sophisticated but practically burdensome alternatives—a pragmatic solution to the longstanding working memory challenge. In this sense, prompt-induced chain-of-thought can be understood as a form of pseudo-symbolic processing: a mechanism that introduces a limited but tangible degree of symbolism into an otherwise fully connectionist system, externalizing intermediate states into the context window, so that subsequent steps can attend to them selectively. This convergence between inference-time prompting and architectural decomposition suggests the principle may hold more broadly; which form of externalization is optimal, under what conditions extensions help rather than harm, and how these effects interact with

model scale are taken up in the following section (Sec. 6.4.5).

6.4.5 Future Work: Diagnostic Methodologies for LLM-Based Abstract Reasoning

The working memory experiment in the preceding section (Sec. 6.4.4) establishes that intermediate reasoning generation is the decisive factor in LLM-based abstract reasoning: prompt engineering and declarative context alone yield essentially no improvement, while any form of step-by-step intermediate generation produces a gain of nearly 50 percentage points at the 4B and 27B scales, though the 0.8B model shows no such benefit, suggesting a minimum capacity threshold below which the mechanism does not activate. However, establishing that the mechanism matters at capable scales leaves the practically important questions open. Which structural features of prompting determine whether it is exploited successfully? At what level of task complexity does it break down? Does the generated reasoning trace reflect genuine abstract reasoning, or is it a brittle mirage: a structurally plausible imitation that disintegrates under distribution shift, as discussed in Sec. 2.3.2 (Zhao et al., 2025; Shojaee et al., 2025)? Answering these questions requires moving beyond aggregate accuracy toward systematic diagnostic methodologies that can isolate the specific components: visual encoding, pattern recognition, logical inference, and answer selection. These components determine whether step-by-step reasoning succeeds or fails on a given instance.

We propose an iterative diagnostic cycle consisting of three stages: *controlled probe generation* to locate capability boundaries, *mechanistic failure diagnosis* to identify why the model fails at those boundaries, and *targeted refinement* guided by the diagnosis. Each stage informs the next, making the cycle self-correcting rather than a one-shot intervention.

Stage 1: Controlled Probe Generation

The first stage locates the precise difficulty level at which model performance transitions from consistent success to systematic failure. We combine LLM generation with symbolic validation: a language model proposes puzzle variants while a symbolic solver verifies rule consistency and answer uniqueness, producing probes guaranteed to be valid RPM instances. Probe difficulty is controlled by extending the standard **I-RAVEN** configuration along targeted axes—scaling matrix dimensions and attribute value ranges beyond standard bounds (as demonstrated by I-RAVEN-X (Hersche et al., 2026)), combining multiple rule types simultaneously, varying the number of panels to be predicted, or introducing out-of-distribution attribute values—with axes chosen based on what the diagnostic stage identifies as failure-inducing. A curriculum approach starts from instances the model solves reliably and progressively increases a single controlled factor, placing probes precisely at the capability boundary and generating minimal pairs that differ only in the factor under investigation.

Stage 2: Failure Diagnosis

The second stage determines *why* performance breaks down at the identified boundary. Recent advances in mechanistic interpretability for large language models (Templeton et al., 2024; Bricken et al., 2023) enable tracing internal representations and discovering circuits associated with specific reasoning operations, crucially distinguishing whether improvements reflect genuine rule induction or statistical shortcutting. Structured prompts that decompose reasoning into explicit stages produce interpretable traces that facilitate stage-wise failure diagnosis, consistent with the decomposition principle underlying the staged self-supervision framework (Sec. 4.2). These methods are most informative at the capability boundaries identified in Stage 1: comparing internal representations on minimally perturbed pairs isolates precisely which features or compositional patterns exceed the model's reasoning capacity. Process-based

evaluation complements interpretability: binary accuracy cannot distinguish near-correct reasoning from random guessing, but per-attribute accuracy and explicit reasoning traces can reveal whether the model correctly identifies individual attribute rules yet fails to integrate them into a coherent prediction, or abandons a correct partial deduction midway through a reasoning chain.

Stage 3: Targeted Refinement and Iteration

The third stage uses the failure diagnosis to guide targeted intervention. If interpretability reveals that the model correctly describes individual panels but fails to identify cross-panel progressions, prompt refinement adds explicit instructions for that transition, provides worked examples of the specific failing step, or restructures the reasoning template around it. Where the failure pattern suggests architectural rather than prompting limitations—for instance, insufficient working memory capacity or inability to compose multiple simultaneous rules—the intervention may instead involve fine-tuning, tool augmentation, or model scaling. Interpretability analysis is then reapplied to verify that internal patterns changed as intended and that improvements generalize beyond the original failure cases. The confirmed diagnosis feeds back into Stage 1, directing probe generation toward the next capability boundary.

Critically, not every failure is expected to be remediable. When a capability boundary persists across prompt refinement, architectural adaptation, and scale increase, the cycle yields a negative result that is nonetheless informative: it delineates what current LLM architectures can and cannot solve under best-effort conditions, providing empirical grounding for claims about reasoning limitations rather than incidental prompt sensitivity. This cycle progressively addresses identifiable bottlenecks through targeted interventions rather than undirected trial-and-error, and provides a principled path toward understanding both the capabilities and the hard limits of LLM-based abstract reasoning systems.

Chapter 7

Summary and Conclusions

This thesis developed and validated a staged self-supervision framework for solving Raven's Progressive Matrices. By decomposing RPM solving into self-supervised property prediction via the **Abstract Compositional Transformer (ACT)** and structured choice making, the framework ranks among the best results reported on RAVEN and I-RAVEN while remaining resistant to dataset biases. This chapter synthesizes the experimental findings, evaluates all three research hypotheses, and reflects on the broader implications of the work.

7.1 Validation of Research Hypotheses

Whether neural networks can learn true abstract reasoning—extracting underlying rules and applying them to configurations unseen during training—remains a central question in the field. High benchmark accuracy does not always reflect this capability: models can achieve strong in-distribution performance by fitting statistical patterns in the training data rather than acquiring transferable reasoning strategies. In RPM benchmarks, this limitation is particularly visible, where the answer set bias (Sec. 3.2.2) allows models to identify the correct answer without interpreting the context panels. The staged self-supervision framework was designed to address this directly: by requiring the model to predict panel properties from context alone, it is compelled to engage with the underlying rules rather than the surface statistics of the answer set. Three hypotheses shaped this design: staged decomposition enables better solvers (**H1**), property prediction mitigates dataset biases (**H2**), and self-supervised learning provides adequate training (**H3**) (Sec. 1.2). The following subsections evaluate each against the experimental evidence.

7.1.1 H1: Staged Decomposition Facilitates Better Solvers

This hypothesis posits that decomposing RPM solving into property prediction and choice making produces superior performance compared to monolithic approaches. The decomposition enables specializa-

tion: representation learning occurs independently of decision-making, allowing each component to focus on its distinct objective, while joint fine-tuning maintains cooperation by refining property representations based on choice-making feedback. Crucially, the decomposition enables the self-supervised training regime evaluated in **H3** (Sec. 1.2). Monolithic approaches couple the query panel to the answer set, restricting masked prediction to a single privileged position. Separating property prediction from choice making allows masking to be applied uniformly across all nine panels—making self-supervised training straightforward where it would otherwise require substantial modeling effort (Sec. 4.3).

The framework achieves **98.27% accuracy on RAVEN**, **97.14% on I-RAVEN**, and **99.30% on RAVEN-FAIR** (Sec. 5.5), placing it among the strongest results reported on these benchmarks and generalizing successfully across all three datasets.

Data efficiency experiments examine what happens when each stage of the decomposition is trained on reduced data, using subsets of 5%, 25%, and 50% of the training set (Sec. 5.2.6). When LCM alone is data-constrained but ACT is fully trained (FR), performance remains near-optimal across all fractions: the core reasoning capability is already encoded in the property representations, and choice making requires comparatively little data to learn on top of them. The reverse is not true—when ACT is data-constrained (RF, RR), no amount of choice-making data recovers the lost reasoning capability, identifying self-supervised property prediction as the primary locus of the framework's reasoning capability. A separate training regime reinforces this conclusion: initializing the entire ACT+LCM architecture randomly and training end-to-end without any masked property prediction objective (Sec. 5.2.1) collapses accuracy to 28% for the Task tokenizer and 34% for the Row tokenizer, substantially below most RF and RR configurations and on par only with the most severely data-constrained setting of 5% data for both stages. This demonstrates that staged decomposition provides inductive structure that substantially improves performance over end-to-end optimization alone.

DCM optimistic estimates analysis provides unexpected further confirmation. An optimistic estimate computes the accuracy achievable when answer panels are compared using ground-truth property vectors rather than learned representations (Sec. 5.1.2); exceeding it would mean that learned distributions are more useful for comparison than ground truth itself. Task+Combined DCM does exactly this (96.97% actual versus 95.39% optimistic), showing that ACT learns to produce property distributions specifically tailored for similarity-based comparison, enabling matching that outperforms ground-truth vectors.

Component specialization through LCM joint fine-tuning reveals a subtler benefit of decomposition. The DCM Row+Combined model is the strongest property predictor in the framework (77.58% *Correct* and a high optimistic estimate of 95.90%), yet its actual choice accuracy of 82.84% falls far below that bound (Sec. 5.1.2). The gap traces to a particular failure mode: 14.94% of incorrect answer panels are misclassified with properties matching the correct answer (Sec. 5.1.3), causing the fixed similarity metric to confuse distractors with the true solution. LCM joint fine-tuning resolves this without any explicit property-prediction supervision. Trained solely on choice-making objectives, it raises incorrect-answer classification from 54.96% to 67.23% and reduces the wrong-matches-correct rate to 0.39% (Sec. 5.2.3); comprehensive analyses and visualizations confirm that learned LCM representations adapt to distractor ambiguities that the fixed DCM metric cannot handle (Sec. 5.1.3).

Taken together, these experiments establish that staged decomposition improves abstract reasoning performance. The competitive results, the data efficiency advantages, and the component specialization achieved through joint training all demonstrate that explicit task structure enables more capable solvers than monolithic approaches.

7.1.2 H2: Property Prediction Mitigates Dataset Biases

With staged decomposition established as beneficial, the question becomes why it succeeds where monolithic approaches fail. Datasets contain exploitable statistical regularities that allow high performance without real understanding; previous work demonstrated that models can solve RPM tasks by analyzing answer set distributions alone, ignoring context panels entirely (Sec. 3.2.2). The hypothesis posits that training models to predict panel properties prevents these shortcuts and compels the development of bias-resistant reasoning strategies.

The framework enforces this through **architectural isolation** at every component (Sec. 5.4). During property prediction, ACT predicts properties of masked panels without accessing answer sets, blocking answer set bias at the representation learning stage itself. Choice-making modules inherit this resistance through strict structural constraints: DCM operates entirely at inference time—ACT parameters are used directly without any dedicated training phase for the choice step—applying a fixed cross-entropy similarity metric, while LCM and CCM process each answer candidate independently in pairwise comparisons. Critically, aggregation across candidates occurs only in the loss function, rather than through trainable components that could discover and exploit distributional patterns, leaving no architectural pathway for the model to compute statistics over the full answer set or learn positional biases.

Cross-dataset generalization provides the clearest experimental confirmation. All models were trained exclusively on RAVEN and evaluated on I-RAVEN and RAVEN-FAIR without retraining; both transfer benchmarks redesigned answer generation specifically to remove the exploitable patterns present in RAVEN (Sec. 3.2.3), so any reliance on RAVEN-specific statistics would cause an immediate performance collapse. Instead, all models achieve results on I-RAVEN and RAVEN-FAIR comparable to those on RAVEN (Sec. 5.1.2, Sec. 5.3.2).

The remaining experiments collectively reinforce this conclusion. Classification and prediction analyses, together with visualizations (Sec. 5.1.3), confirm that models consistently organize their internal representations around panel properties rather than answer set statistics. Most strikingly, this holds even for LCM, which carries no explicit property prediction objective: as established in the component specialization analysis above, LCM trained solely on choice-making still markedly improves incorrect-answer classification (Sec. 5.2.3). A model exploiting biases would have no incentive to refine property-based representations in this way; the fact that it does so confirms that staged decomposition compels property-based reasoning even when not explicitly required by the training objective.

Cross-dataset transfer, architectural isolation, and the mechanistic classification evidence collectively establish that property prediction successfully mitigates dataset biases. Rather than relying on dataset cleaning or adversarial training, the framework achieves bias resistance through task reformulation that eliminates the opportunity for shortcut learning at its source, while the resulting emphasis on property-based reasoning also reduces the utility of shortcuts that do not rely on answer set statistics.

7.1.3 H3: Self-Supervision Provides Adequate Training

Having established that staged decomposition works and that it mitigates biases, the final question concerns the training methodology that makes the preceding results possible. Monolithic end-to-end approaches train directly on the answer selection goal, creating dependence on answer sets and tying the learned representations to that objective. The hypothesis posits that self-supervised learning through masked panel property prediction provides sufficient signal for developing robust reasoning capabilities.

Strong benchmark results confirm self-supervision adequacy. ACT achieves 98.26–98.27% accuracy on RAVEN through self-supervised property prediction followed by choice-making training (Sec. 5.5), while training the complete architecture from scratch in end-to-end fashion (Fresh) collapses to 28–34% choice accuracy (Sec. 5.2.1) despite identical model capacity. This 64–70 percentage-point

gap demonstrates that self-supervised property prediction provides the critical training signal: model capacity alone proves insufficient without appropriate training methodology.

Masking strategy quality determines how much of that capability is realized. For the Task tokenizer, random masking improves property prediction from 18.91% to 72.63% *Correct* compared to query-only masking (Sec. 4.5), because predicting any masked panel from diverse contexts functions as both data augmentation and regularization, producing representations robust to position-specific biases. The importance of this choice becomes extreme at the downstream level: Task+Query, trained with query-only masking, achieves 5.44% choice accuracy, while the identical Task tokenizer architecture trained with random masking reaches 96.97%, exceeding its 95.39% optimistic estimate and representing a 91.53 percentage-point transformation from the same model trained differently (Sec. 5.1.2).

Architectural prerequisites explain why the transformer is essential for realizing these gains. Without attention mechanisms, the model cannot relate information across panels, and diverse masking contexts carry no useful signal: replacing the transformer with a feedforward architecture of matched parameter count and computational budget (DenseFormer) reduces property prediction to below 1% *Correct* (Sec. 4.6.2). With attention, any panel can inform predictions about any other through learned dependencies, which is what makes random masking informationally rich rather than merely noisy. The representations that emerge from this combination are not opaque: t-SNE visualizations reveal clear clustering by arrangement type (Sec. 4.8.1, Fig. 4.10), and SHAP attribution identifies sparse, localized token dimensions encoding specific properties (Sec. 4.8.2, Fig. 4.11), suggesting that self-supervised property prediction encourages compositional, symbolic-like representation.

Data efficiency and emergent behavior provide final confirmation. As established in **H1**, data efficiency experiments (Sec. 5.2.6) reveal that self-supervised property prediction constitutes the core reasoning capability: training ACT on reduced data substantially degrades performance, while choice-making training on reduced data causes minimal degradation. Beyond this, LCM receives only the correct answer index as supervision (no panel property signal whatsoever), yet through choice-making optimization it still substantially improves incorrect-answer classification, essentially eliminating the systematic failure mode (Sec. 5.2.3). The self-supervised representations prove sufficiently dependable that choice-making training refines and enhances them rather than replacing them with entirely new representations.

The evidence from benchmark performance, masking strategy ablations, architectural comparisons, and data efficiency analysis consistently points to the same conclusion: self-supervision provides adequate training for abstract reasoning when the masking strategy creates informationally rich prediction contexts and the architecture is equipped to exploit them. Catastrophic failures under inadequate configurations (query-only masking and feedforward architectures) illustrate the necessity of both components. Models trained with proper self-supervision discover emergent conceptual structure beyond their explicit training signal and transfer these capabilities to downstream tasks, confirming substantive rather than superficial learning. The staged framework enables this: decomposing the task as posited in **H1** (Sec. 1.2) allows random masking to be applied freely across all panel positions, while architectural isolation from answer sets as required by **H2** (Sec. 1.2) ensures the learned representations remain grounded in panel properties rather than dataset statistics.

7.2 Contributions and Impact

The staged self-supervision framework introduced in this thesis demonstrates that decomposing a visual reasoning task into self-supervised property prediction and choice making yields both stronger performance and greater robustness than treating the task as a single end-to-end objective. The ACT achieves this through a training regime that requires the model to infer panel properties from context alone, which naturally enforces property-based reasoning and generalizes across benchmark variants designed to

expose shortcut exploitation. This work contributes to bridging subsymbolic learning with symbolic representations, showing that abstract reasoning benefits from moving beyond pure connectionism. Success stems from the coherent integration of task decomposition, architectural constraints, and self-supervised learning—three principles that reinforce one another rather than operating independently. Scale amplifies rather than substitutes for this structure: transformers provide the capacity to learn intricate cross-panel dependencies, but without careful design, increased capacity merely enables more sophisticated pattern matching. Future extensions combining these principles with automated symbolic discovery mechanisms could enable broader generalization across abstract reasoning domains without manual property engineering. Effective abstract reasoning requires principled design integrated with scale, providing a foundation for AI systems that discover genuine compositional patterns rather than exploiting statistical shortcuts.

Bibliography

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, et al. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/arXiv.2303.08774. 181 additional authors not shown.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL `https://arxiv.org/abs/2305.13245`.

Mohammad Mahmudul Alam, Alexander Oberle, Edward Raff, Stella Biderman, Tim Oates, and James Holt. Walsh-hadamard variational quantum circuit for classification. 2024.

Luciano Alparone, Lucien Wald, Jocelyn Chanussot, Claire Thomas, Paolo Gamba, and Lorenzo Milli Bruce. A global quality measurement of pan-sharpened multispectral imagery. *IEEE Geoscience and Remote Sensing Letters*, 1(4):313–317, 2004.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety, 2016. URL `https://arxiv.org/abs/1606.06565`.

Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. Technical Report SNUDM-TR-2015-03, Seoul National University, Department of Industrial Engineering, 2015. URL `http://dm.snu.ac.kr/static/docs/TR/SNUDM-TR-2015-03.pdf`.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, 2016.

Fazel Arasteh, Mohammed Elmahgiubi, Behzad Khamidehi, Hamidreza Mirkhani, Weize Zhang, Tongtong Cao, and Kasra Rezaee. Validity learning on failures: Mitigating the distribution shift in autonomous vehicle planning, 2024. URL `https://arxiv.org/abs/2406.01544`.

Yannis Assael, Thea Sommerschield, Brendan Shillingford, Mahyar Bordbar, John Pavlopoulos, Marita Chatzipanagiotou, Ion Androutsopoulos, Jonathan Prag, and Nando de Freitas. Restoring and attributing ancient texts using deep neural networks. *Nature*, 603(7900):280–283, 2022. URL `https://doi.org/10.1038/s41586-022-04448-z`.

Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples, 2018. URL `https://arxiv.org/abs/1802.00420`.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, jul 2016. URL https://arxiv.org/abs/1607.06450.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic generalization: What is required and can it be learned?, 2018. URL https://arxiv.org/abs/1811.12889.

David Barrett, Felix Hill, Adam Santoro, Ari Morcos, and Timothy Lillicrap. Measuring abstract reasoning in neural networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 511–520. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/barrett18a.html.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Jakub Bednarek and Krzysztof Krawiec. Learning to solve abstract reasoning problems with neurosymbolic program synthesis and task generation. In *18th International Conference on Neural-Symbolic Learning and Reasoning*, pages 321–339. Springer, 2024. URL https://link.springer.com/chapter/10.1007/978-3-031-71167-1_21.

Sara Beery, Grant Van Horn, and Pietro Perona. Recognition in terra incognita. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 456–473, 2018. URL https://openaccess.thecvf.com/content_ECCV_2018/html/Beery_Recognition_in_Terra_ECCV_2018_paper.html.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48, 2009. URL https://doi.org/10.1145/1553374.1553380.

Yaniv Benny, Niv Pekar, and Lior Wolf. Scale-localized abstract reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12552–12560, 2021. doi: 10.1109/CVPR46437.2021.01237. URL https://doi.org/10.1109/CVPR46437.2021.01237.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, pages 2546–2554, 2011.

Leonardo Berti, Flavio Giorgi, and Gjergji Kasneci. Emergent abilities in large language models: A survey, 2025. URL https://arxiv.org/abs/2503.05788.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu

Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models, 2021. URL `https://arxiv.org/abs/2108.07258`.

Mikhail M. Bongard. *Pattern Recognition*. Spartan Books, Rochelle Park, N.J., 1970.

Gwern Branwen. The neural net tank urban legend, 2020. URL `https://gwern.net/tank`.

Wieland Brendel and Matthias Bethge. Approximating CNNs with bag-of-local-features models works surprisingly well on ImageNet, 2019. URL `https://arxiv.org/abs/1904.00760`.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Chris Olah. Towards monosemanticity: Decomposing language models with dictionary learning. Transformer Circuits Thread, October 2023. URL `https://transformer-circuits.pub/2023/monosemantic-features`.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html`.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiences with GPT-4, 2023. URL `https://arxiv.org/abs/2303.12712`.

Ethan Caballero, Kshitij Gupta, Irina Rish, and David Krueger. Broken neural scaling laws, 2022. URL `https://arxiv.org/abs/2210.14891`.

Giacomo Camposampiero, Michael Hersche, Aleksandar Terzić, Roger Wattenhofer, Abu Sebastian, and Abbas Rahimi. Towards learning abductive reasoning using VSA distributed representations. In *18th International Conference on Neural-Symbolic Learning and Reasoning (NeSy)*, September 2024.

Giacomo Camposampiero, Michael Hersche, Roger Wattenhofer, Abu Sebastian, and Abbas Rahimi. Can large reasoning models do analogical reasoning under perceptual uncertainty? In *Proceedings of the 19th International Conference on Neurosymbolic Learning and Reasoning (NeSy)*, volume 284 of *Proceedings of Machine Learning Research*, pages 750–776, 2025. URL `https://proceedings.mlr.press/v284/camposampiero25a.html`.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2017. URL `https://doi.org/10.1109/SP.2017.49`.

Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness, 2019. URL `https://arxiv.org/abs/1902.06705`.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models, 2023. URL `https://arxiv.org/abs/2202.07646`.

Patricia A Carpenter, Marcel A Just, and Peter Shell. What one intelligence test measures: a theoretical account of the processing in the Raven progressive matrices test. *Psychological Review*, 97(3):404–431, 1990. doi: 10.1037/0033-295X.97.3.404. URL https://pubmed.ncbi.nlm.nih.gov/2381998/.

Minshuo Chen, Yu Bai, Jason D Lee, Tuo Zhao, Huan Wang, Caiming Xiong, and Richard Socher. Towards understanding hierarchical learning: Benefits of neural representations, 2020a. URL https://arxiv.org/abs/2006.13436.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607. PMLR, 2020b. URL https://arxiv.org/abs/2002.05709.

Xi Chen, Zhihong Deng, Gehui Shen, and Ting Huang. A novel framework for recurrent neural networks with enhancing information processing and transmission between units, 2018. URL https://arxiv.org/abs/1806.00628.

François Chollet. On the measure of intelligence, 2019. URL https://arxiv.org/abs/1911.01547.

François Chollet. ARC-AGI-2: The next frontier in artificial general intelligence, 2025.

François Chollet. How i think about LLM prompt engineering. *Sparks in the Wind*, October 2023. URL https://fchollet.substack.com/p/how-i-think-about-llm-prompt-engineering.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. PaLM: Scaling language modeling with pathways, 2022. URL https://arxiv.org/abs/2204.02311.

Jessica B. Cicchino. What humanlike errors do autonomous vehicles need to avoid to maximize safety? *Journal of Safety Research*, 75:310–318, 2020. URL https://pubmed.ncbi.nlm.nih.gov/33334489/.

Felipe Codevilla, Eder Santana, Antonio M. Lopez, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, 2019. URL https://openaccess.thecvf.com/content_ICCV_2019/papers/Codevilla_Exploring_the_Limitations_of_Behavior_Cloning_for_Autonomous_Driving_ICCV_2019_paper.pdf.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing, 2019. URL https://arxiv.org/abs/1902.02918.

Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Neisarg Dave, Daniel Kifer, C. Lee Giles, and Ankur Mali. Investigating symbolic capabilities of large language models, 2024. URL https://arxiv.org/abs/2405.13209.

Artur d'Avila Garcez and Luís C. Lamb. Neurosymbolic AI: The 3rd wave. *Artificial Intelligence Review*, 56 (11):12387–12406, 2023. doi: 10.1007/s10462-023-10448-w.

Nando de Freitas. It's all about scale now! the game is over! Twitter, May 2022. URL `https://x.com/NandoDF/status/1525397036325019649`.

DeepSeek-AI. DeepSeek-V3 technical report, 2024. URL `https://arxiv.org/abs/2412.19437`.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *CoRR*, abs/2501.12948, 2025. doi: 10.48550/arXiv.2501.12948.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022. URL `https://doi.org/10.1038/s41586-021-04301-9`.

Chunyuan Deng, Yilun Zhao, Xiangru Tang, Mark Gerstein, and Arman Cohan. Investigating data contamination in modern benchmarks for large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics*, 2024. URL `https://arxiv.org/abs/2311.09783`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional Transformers for language understanding, 2018. URL `https://arxiv.org/abs/1810.04805`.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, Oct 2020. URL `http://arxiv.org/abs/2010.11929`.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of Transformers on compositionality, 2023. URL `https://arxiv.org/abs/2305.18654`.

Yanai Elazar, Nora Kassner, Shauli Ravfogel, Amir Feder, Abhilasha Ravichander, Marius Mosbach, Yonatan Belinkov, Hinrich Schütze, and Yoav Goldberg. Measuring causal effects of data statistics on language model "factual" predictions, 2022. URL `https://arxiv.org/abs/2207.14251`.

Kevin Ellis, Catherine Wong, Maxwell Nye, and Mary Sablé. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. *arXiv preprint arXiv:2006.08381*, 2020.

Thomas G. Evans. A heuristic program to solve geometric-analogy problems. 1964.

François Fleuret, Ting Li, Charles Dubout, Emma K. Wampler, Steven Yantis, and Donald Geman. Comparing machines and humans on a visual categorization test. *Proceedings of the National Academy of Sciences*, 108 (43):17621–17625, 2011. doi: 10.1073/pnas.1109168108.

Jerry A Fodor and Zenon W Pylyshyn. *Connectionism and Cognitive Architecture.* 1988.

Marta Garnelo and Murray Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17–23, 2019. doi: 10.1016/j.cobeha.2018.12.010.

Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning, 2016. URL https://arxiv.org/abs/1609.05518.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016. URL https://doi.org/10.1109/CVPR.2016.265.

Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bygh9j09KX.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11): 665–673, 2020. URL https://doi.org/10.1038/s42256-020-00257-z.

Gemini Team, Google. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805, 2023. doi: 10.48550/arXiv.2312.11805.

Gaël Gendron, Qiming Bao, Michael Witbrock, and Gillian Dobbie. Large language models are not strong abstract reasoners, 2023. URL https://arxiv.org/abs/2305.19555.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories, 2021. URL https://arxiv.org/abs/2012.14913.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014. URL https://arxiv.org/abs/1412.6572.

Google. Bard: A large language model based chatbot. https://blog.google/technology/ai/try-bard/, 2023. Google AI Blog, March 21, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The Llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/arXiv.2407.21783. 460 additional authors not shown.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. URL https://arxiv.org/abs/1410.5401.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016. URL https://doi.org/10.1038/nature20101.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 21271–21284, 2020. URL https://arxiv.org/abs/2006.07733.

Kathryn Hamilton, Ankita Nayak, Božidar Božić, and Luca Longo. Neuro symbolic ai: The 3rd wave. *Artificial Intelligence Review*, 56(11):12387–12569, 2023. URL https://doi.org/10.1007/s10462-023-10448-w.

Demis Hassabis and Dwarkesh Patel. Demis hassabis — scaling, superhuman ais, AlphaZero atop LLMs, AlphaFold. Podcast interview, 2024. URL https://www.dwarkesh.com/p/demis-hassabis.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009. URL https://web.stanford.edu/~hastie/ElemStatLearn/.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. URL https://doi.org/10.1109/CVPR.2016.90.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9729–9738, 2020. URL https://arxiv.org/abs/1911.05722.

Dan Hendrycks and Kevin Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv*, abs/1606.08415, 2016.

Michael Hersche, Francesco di Stefano, Thomas Hofmann, Abu Sebastian, and Abbas Rahimi. Probabilistic abduction for visual abstract reasoning via learning rules in vector-symbolic architectures. In *3rd Workshop on Math-AI (MATH-AI@NeurIPS)*, 2023a.

Michael Hersche, Geethan Karunaratne, Giovanni Cherubini, Luca Benini, Abu Sebastian, and Abbas Rahimi. A neuro-vector-symbolic architecture for solving Raven's progressive matrices. *Nature Machine Intelligence*, 5 (4):363–375, 2023b. URL https://doi.org/10.1038/s42256-023-00630-8.

Michael Hersche, Giacomo Camposampiero, Roger Wattenhofer, Abu Sebastian, and Abbas Rahimi. Towards learning to reason: Comparing LLMs with neuro-symbolic on arithmetic relations in abstract reasoning. *Neurosymbolic Artificial Intelligence*, 2:29498732261419316, 2026. doi: 10.1177/29498732261419316. URL https://doi.org/10.1177/29498732261419316.

Irina Higgins, Loic Matthey, Arka Pal, Xavier Glorot, Benigno Uria, Charles Blundell, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Sy2fzU9gl.

Felix Hill, Adam Santoro, David G.T. Barrett, Ari S. Morcos, and Timothy Lillicrap. Learning to make analogies by contrasting abstract relational structure. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=SylLYsCcFm.

Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012. URL https://doi.org/10.1109/MSP.2012.2205597.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Douglas R. Hofstadter. *Fluid concepts & creative analogies : computer models of the fundamental mechanisms of thought*. Basic Books, New York, 1995. ISBN 0465051545.

John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. *Induction: Processes of inference, learning, and discovery*. MIT Press, 1986.

Sheng Hu, Yuqing Ma, Xianglong Liu, Yanlu Wei, and Shihao Bai. Stratified rule-aware network for abstract visual reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 35, pages 1567–1574, 2021.

Xiaoyang Hu, Shane Storks, Richard L. Lewis, and Joyce Chai. In-context analogical reasoning with pre-trained language models, 2023a. URL https://arxiv.org/abs/2305.17626.

Xuemin Hu, Shen Li, Tingyu Huang, Bo Tang, Rouxing Huai, and Long Chen. How simulation helps autonomous driving: A survey of sim2real, digital twins, and parallel intelligence, 2023b. URL https://arxiv.org/abs/2305.01263.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models, 2023. URL https://arxiv.org/abs/2311.05232.

Drew A Hudson and Christopher D Manning. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *CVPR*, 2019.

HuggingFace M4 Team. IDEFICS-2 fine-tuned on RAVEN progressive matrices. Hugging Face Model Hub, 2024. URL https://huggingface.co/HuggingFaceM4/idefics2_raven_finetuned.

Rob J Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2nd edition, 2018. URL https://otexts.com/fpp2/.

Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *Advances in Neural Information Processing Systems*, pages 125–136, 2019. URL https://papers.nips.cc/paper/8307-adversarial-examples-are-not-bugs-they-are-features.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1125–1134, 2017. URL https://doi.org/10.1109/CVPR.2017.632.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

Huaizu Jiang, Xiaojian Ma, Weili Nie, Zhiding Yu, Yuke Zhu, and Anima Anandkumar. Bongard-HOI: Benchmarking few-shot visual reasoning for human-object interactions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 19056–19065, 2022. URL https://arxiv.org/abs/2205.13803.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017.

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011. ISBN 978-0374275631. URL https://us.macmillan.com/books/9780374533557/thinkingfastandslow.

Adam Tauman Kalai, Ofir Nachum, Santosh S. Vempala, and Edwin Zhang. Why language models hallucinate. 2025.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey, 2020. URL https://arxiv.org/abs/2006.12057.

Justin Kim, Matthew Ricci, and Thomas Serre. Not-So-CLEVR: learning same–different relations strains feedforward neural networks. *Interface Focus*, 8(4), 2018. URL https://royalsocietypublishing.org/doi/10.1098/rsfs.2018.0011.

Seungmin Kim, Suhyun Ji, Jongyoo Lee, and Seungwan Hong. Self-knowledge distillation with progressive refinement of targets. In *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021. URL https://openaccess.thecvf.com/content/ICCV2021/papers/Kim_Self-Knowledge_Distillation_With_Progressive_Refinement_of_Targets_ICCV_2021_paper.pdf.

Youngsung Kim, Jinwoo Shin, Eunho Yang, and Sung Ju Hwang. Few-shot visual reasoning with meta-analogical contrastive learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16846–16856. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/c39e1a03859f9ee215bc49131d0caf33-Paper.pdf.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2015. URL https://arxiv.org/abs/1412.6980.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. URL https://doi.org/10.1073/pnas.1611835114.

Denis Kleyko, Mike Davies, E. Paxon Frady, Pentti Kanerva, Spencer J. Kent, Bruno A. Olshausen, Evgeny Osipov, Jan M. Rabaey, Dmitri A. Rachkovskij, Abbas Rahimi, and Friedrich T. Sommer. Vector symbolic architectures as a computing framework for emerging hardware. *Proceedings of the IEEE*, 110(10):1538–1571, 2022. doi: 10.1109/JPROC.2022.3209062.

Konstantin Krestnikov. Compression favors consistency, not truth: When and why language models prefer correct information. *arXiv preprint arXiv:2603.11749*, 2026.

Fred A. Kruse, A. B. Lefkoff, J. W. Boardman, K. B. Heidebrecht, A. T. Shapiro, P. J. Barloon, and A. F. H. Goetz. The spectral image processing system (SIPS)—interactive visualization and analysis of imaging spectrometer data. *Remote Sensing of Environment*, 44:145–163, 1993.

Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, 2018. URL https://arxiv.org/abs/1808.06226.

Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. URL https://doi.org/10.1214/aoms/1177729694.

Maithilee Kunda, Keith McGreggor, and Ashok K. Goel. Taking the machine to the vision: A cognitively motivated approach to vision-based artificial intelligence. 2010.

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks, 2018. URL https://arxiv.org/abs/1711.00350.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. URL https://doi.org/10.1126/science.aab3050.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

Federico Landi, Lorenzo Baraldi, Marcella Cornia, and Rita Cucchiara. Working memory connections for LSTM, 2021. URL https://arxiv.org/abs/2109.00020.

Hugo Laurençon, Léo Tronchon, Matthieu Cord, and Victor Sanh. What matters when building vision-language models?, 2024. URL https://arxiv.org/abs/2405.02246.

Yann LeCun. X post endorsing approximate-retrieval view, 2023. URL https://x.com/ylecun/status/1713228046033936717.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. URL https://doi.org/10.1109/5.726791.

Itay Levy, Ben Bogin, and Jonathan Berant. Diverse demonstrations improve in-context compositional generalization. *arXiv preprint arXiv:2212.06800*, 2022.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Pontone, Douwe Kiela, and Sebastian Riedel. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

Yichao Liang. Learning to infer 3d shape programs with differentiable renderer, 2022. URL https://arxiv.org/abs/2206.12675.

Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991. URL https://doi.org/10.1109/18.61115.

Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 89–105, 2018. URL https://doi.org/10.1007/978-3-030-01252-6_6.

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields, 2020. URL https://arxiv.org/abs/2007.11571.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015. URL https://doi.org/10.1109/CVPR.2015.7298965.

Andrew Lovett, Kenneth Forbus, Emmett Tomai, and Jeffrey Usher. Modeling the effect of working memory capacity on analogy problems. 2007.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks, Jan 2017. URL http://arxiv.org/abs/1701.04128.

Peter Xiangyuan Ma, Cherry Ng, Leandro Rizk, Steve Croft, Andrew P.V. Siemion, Bryan Brzycki, Daniel Czech, Jamie Drew, Vishal Gajjar, John Hoang, et al. A deep-learning search for technosignatures of 820 nearby stars, 2023. URL https://arxiv.org/abs/2301.12670.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. ISSN 1533-7928. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020. URL https://doi.org/10.1016/j.ijforecast.2019.04.014.

Mikołaj Małkiński and Jacek Mańdziuk. Deep learning methods for abstract visual reasoning: A survey on Raven's Progressive Matrices. *ACM Comput. Surv.*, 57(7), February 2025. ISSN 0360-0300. doi: 10.1145/3715093. URL https://doi.org/10.1145/3715093.

Gary F. Marcus. Deep learning: A critical appraisal, 2018. URL https://arxiv.org/abs/1801.00631.

Gary F. Marcus. The next decade in ai: Four steps towards robust artificial intelligence, 2020. URL https://arxiv.org/abs/2002.06177.

Mikołaj Małkiński and Jacek Mańdziuk. Multi-label contrastive learning for abstract visual reasoning. *IEEE Transactions on Neural Networks and Learning Systems*, 35(2):1941–1953, 2022. doi: 10.1109/TNNLS.2022.3185949.

Mikołaj Małkiński and Jacek Mańdziuk. A review of emerging research directions in abstract visual reasoning. *Information Fusion*, 91:713–736, 2023. doi: 10.1016/j.inffus.2022.11.011.

Mikołaj Małkiński and Jacek Mańdziuk. One Self-Configurable Model to Solve Many Abstract Visual Reasoning Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(13):14297–14305, March 2024. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v38i13.29342. URL https://ojs.aaai.org/index.php/AAAI/article/view/29342.

Mikołaj Małkiński and Jacek Mańdziuk. A-I-RAVEN and I-RAVEN-mesh: Two new benchmarks for abstract visual reasoning. In James Kwok, editor, *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 5932–5940. International Joint Conferences on Artificial Intelligence Organization, 8 2025a. doi: 10.24963/ijcai.2025/660. URL https://doi.org/10.24963/ijcai.2025/660. Main Track.

Mikołaj Małkiński and Jacek Mańdziuk. Advancing generalization across a variety of abstract visual reasoning tasks. In James Kwok, editor, *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pages 5941–5949. International Joint Conferences on Artificial Intelligence Organization, 8 2025b. doi: 10.24963/ijcai.2025/661. URL https://doi.org/10.24963/ijcai.2025/661. Main Track.

Jacek Mańdziuk and Adam Żychowski. Deepiq: A human-inspired ai system for solving iq test problems. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019. doi: 10.1109/IJCNN.2019.8851878.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989. URL https://doi.org/10.1016/S0079-7421(08)60536-8.

Tom McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 2013. URL https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, and Samy Bengio. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024. URL https://arxiv.org/abs/2410.05229.

Melanie Mitchell. Abstraction and analogy-making in artificial intelligence. *Annals of the New York Academy of Sciences*, 1505(1):79–101, 2021. URL https://doi.org/10.1111/nyas.14619.

Melanie Mitchell. Evaluating large language models using "counterfactual tasks". AI Guide (Substack), 2024. URL https://aiguide.substack.com/p/evaluating-large-language-models.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

Son The Nguyen, Theja Tulabandhula, and Mary Beth Watson-Manheim. User friendly and adaptable discriminative ai, 2024. URL https://arxiv.org/abs/2312.06826.

Weili Nie, Zhiding Yu, Lei Mao, Ankit B. Patel, Yuke Zhu, and Animashree Anandkumar. Bongard-LOGO: A new benchmark for human-level concept learning and reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/bf15e9bbff22c7719020f9df4badc20a-Abstract.html.

Timothy Niven and Hung-Yu Kao. Probing neural network comprehension of natural language arguments. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4658–4664, 2019. URL https://doi.org/10.18653/v1/P19-1459.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Gpt3-to-plan: Extracting plans from text using GPT-3. In *ICAPS 2021 Workshop on Planning for Financial Services*, 2021. URL https://arxiv.org/abs/2106.07131.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads, 2022. URL https://arxiv.org/abs/2209.11895.

OpenAI. OpenAI o3-mini system card. Technical report, OpenAI, 2025. URL https://openai.com/index/openai-o3-mini-system-card.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2009. URL https://doi.org/10.1109/TKDE.2009.191.

Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks, 2016a. URL https://arxiv.org/abs/1605.07277.

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597, 2016b. URL https://doi.org/10.1109/SP.2016.41.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, 2002. URL https://aclanthology.org/P02-1040/.

Peter S Park, Philipp Schoenegger, Chongyang Zhu, Max Levendusky, Amy A Winecoff, Arthur Juliani, and Chris Piech. The generative ai paradox: "what it can create, it may not understand", 2023. URL https://arxiv.org/abs/2311.00059.

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2536–2544, 2016. URL https://doi.org/10.1109/CVPR.2016.278.

Zhiliang Peng, Li Dong, Hangbo Bao, Qiang Liu, and Furu Wei. BEiT v2: Masked image modeling with vector-quantized visual tokenizers, 2022. URL https://arxiv.org/abs/2208.06366.

Felix Petersen, Bastian Goldluecke, Christian Borgelt, and Oliver Deussen. Gendr: A generalized differentiable renderer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4994–5003, June 2022. URL https://openaccess.thecvf.com/content/CVPR2022/papers/Petersen_GenDR_A_Generalized_Differentiable_Renderer_CVPR_2022_paper.pdf.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. Hypothesis only baselines in natural language inference. *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 180–191, 2018. URL https://doi.org/10.18653/v1/S18-2023.

Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Weize Chen, Yusheng Su, Yufan Dang, Jiahao Li, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development, 2023. URL https://arxiv.org/abs/2307.07924.

Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, 2026.

Dmitri A Rachkovskij and Ernst M Kussul. Representation and processing of structures with binary sparse distributed codes. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):261–276, 2001. URL https://doi.org/10.1109/69.917565.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

John C. Raven. Mental tests used in genetic studies: The performance of related individuals on tests mainly eductive and mainly reproductive. Master's thesis, University of London, London, 1936.

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet?, 2019. URL https://arxiv.org/abs/1902.10811.

Edmar Rezende, Guilherme Ruppert, Tiago Carvalho, Fábio Ramos, and Paulo de Geus. Malicious software classification using transfer learning of ResNet-50 deep neural network. In *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1011–1014. IEEE, 2017.

Matthew Ricci, Justin Kim, and Thomas Serre. Same-different problems strain convolutional neural networks, 2018. URL https://arxiv.org/abs/1802.03390.

Joshua Robinson, Ching-Yao Chuang, Suvrit Sra, and Stefanie Jegelka. Contrastive learning with hard negative samples. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.04592.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in BERTology: What we know about how BERT works. *TACL*, 8:842–866, 2020. URL https://doi.org/10.1162/tacl_a_00349.

Pedro Elias Ruiz. Building and solving odd-one-out classification problems: A systematic approach. *Intelligence*, 39(5):342–350, 2011. doi: 10.1016/j.intell.2011.06.002.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016. URL https://arxiv.org/abs/1606.04671.

Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications, 2024. URL https://arxiv.org/abs/2402.07927.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 4967–4976. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/e6acf4b0f69f6f6e60e9a815938aa1ff-Paper.pdf.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage?, 2023. URL https://arxiv.org/abs/2304.15004.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016. URL https://arxiv.org/abs/1508.07909.

Steven Shaw, Denis Kleyko, E Paxon Frady, Friedrich T Sommer, and Bruno A Olshausen. Developing a foundation of vector symbolic architectures using category theory, 2025. URL https://arxiv.org/abs/2501.05368.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL https://arxiv.org/abs/1701.06538.

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity, 2025. URL https://arxiv.org/abs/2506.06941.

Carl-Johann Simon-Gabriel, Yann Ollivier, Leon Bottou, Bernhard Schölkopf, and David Lopez-Paz. First-order adversarial vulnerability of neural networks and input dimension. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5809–5817. PMLR, 2019. URL https://proceedings.mlr.press/v97/simon-gabriel19a.html.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S. Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, Perry Payne, Martin Seneviratne, Paul Gamble, Chris Kelly, Nathanael Schärli, Aakanksha Chowdhery, Philip Mansfield, Blaise Agüera y Arcas, Dale Webster, and Greg S. Corrado. Large language models encode clinical knowledge. *arXiv preprint arXiv:2212.13138*, 2022.

Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Casper Kaae Sønderby, Lasse Espeholt, Jonathan Heek, Mostafa Dehghani, Avital Oliver, Tim Salimans, Shreya Agrawal, Jason Hickey, and Nal Kalchbrenner. MetNet: A neural weather model for precipitation forecasting, 2020. URL https://arxiv.org/abs/2003.12140.

Yu Song, Madhav V. Chitturi, and David A. Noyce. Automated vehicle crash sequences: Patterns and potential uses for crash avoidance, 2020. URL `https://arxiv.org/abs/2102.06286`.

Steven Spratley, Krista Ehinger, and Tim Miller. A closer look at generalisation in RAVEN. In *Computer Vision – ECCV 2020: 16th European Conference on Computer Vision, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX*, volume 12364 of *Lecture Notes in Computer Science*, pages 601–616. Springer, 2020. doi: 10.1007/978-3-030-58529-7_36.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Robert J. Sternberg. *Beyond IQ: A triarchic theory of human intelligence*. Cambridge University Press, 1985.

Claes Strannegård, Mehrdad Amirghasemi, and Simon Ulfsbäcker. Anthropomorphic reasoning about visual analogies. 2013.

Qiao Sun, Huimin Wang, Jiahao Zhan, Fan Nie, Xin Wen, Leimeng Xu, Kun Zhan, Peng Jia, Xianpeng Lang, and Hang Zhao. Generalizing motion planners with mixture of experts for autonomous driving, 2024. URL `https://arxiv.org/abs/2410.15774`.

Zhong-Hua Sun, Ru-Yuan Zhang, Zonglei Zhen, Da-Hui Wang, Yong-Jie Li, Xiaohong Wan, and Hongzhi You. Systematic abductive reasoning via diverse relation representations in vector-symbolic architecture. *arXiv preprint arXiv:2501.11896*, 2025.

Richard S. Sutton. The bitter lesson, 2019. URL `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`.

Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/tan19a.html`.

Mingxing Tan and Quoc V. Le. EfficientNetV2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10096–10106. PMLR, 2021. URL `https://proceedings.mlr.press/v139/tan21a.html`.

Hao Tang and Kevin Ellis. From perception to programs: Regularize, overparameterize, and amortize, 2022. URL `https://arxiv.org/abs/2206.05922`.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C Daniel Freeman, Theodore R Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. Transformer Circuits Thread, May 2024. URL `https://transformer-circuits.pub/2024/scaling-monosemanticity/`.

Giovanni Thoms and Mia Richardson. Solving ARC visual analogies with neural embeddings and vector arithmetic: A generalized method, 2023. URL `https://arxiv.org/abs/2311.08083`.

Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, YaGuang Li, Hongrae Lee, Huaixiu Steven Zheng, Amin Ghafouri, Marcelo Menegali, Yanping Huang, Maxim Krikun, Dmitry Lepikhin, James Qin, Dehao Chen, Yuanzhong Xu, Zhifeng Chen, Adam Roberts, Maarten Bosma, Vincent Zhao, Yanqi Zhou, Chung-Ching Chang, Igor Krivokon, Will Rusch, Marc Pickett, Pranesh Srinivasan, Laichee Man, Kathleen Meier-Hellstern, Meredith Ringel Morris, Tulsee Doshi, Renelito Delos Santos, Toju Duke, Johnny Soraker, Ben Zevenbergen,

Vinodkumar Prabhakaran, Mark Diaz, Ben Hutchinson, Kristen Olson, Alejandra Molina, Erin Hoffman-John, Josh Lee, Lora Aroyo, Ravi Rajakumar, Alena Butryna, Matthew Lamm, Viktoriya Kuzmina, Joe Fenton, Aaron Cohen, Rachel Bernstein, Ray Kurzweil, Blaise Aguera-Arcas, Claire Cui, Marian Croak, Ed Chi, and Quoc Le. LaMDA: Language models for dialog applications, 2022. URL https://arxiv.org/abs/2201.08239.

Simen Thys, Wiebe Van Ranst, and Toon Goedemé. Fooling automated surveillance cameras: Adversarial patches to attack person detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 49–55, 2019.

Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rkZvSe-RZ.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for LLMs on planning and reasoning about change), 2022. URL https://arxiv.org/abs/2206.10498.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1606.01781*, 2017.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. URL https://arxiv.org/abs/1509.06461.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

Lucien Wald. Quality of high resolution synthesised images: Is there a simple criterion? In *Proceedings of the Third Conference Fusion of Earth Data*, pages 99–103, 2000.

Duo Wang, Mateja Jamnik, and Pietro Lio. Abstract Diagrammatic Reasoning with Multiplex Graph Networks, June 2020. URL http://arxiv.org/abs/2006.11197.

Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *Proceedings of the 37th International Conference on Machine Learning*, PMLR 119, pages 9929–9939, 2020. URL https://proceedings.mlr.press/v119/wang20k.html.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. 2022.

Zhou Wang and Alan C. Bovik. A universal image quality index. *IEEE Signal Processing Letters*, 9(3):81–84, 2002.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48, pages 1995–2003, 2016. URL https://arxiv.org/abs/1511.06581.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022a. URL https://arxiv.org/abs/2206.07682.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2022b. URL https://arxiv.org/abs/2201.11903.

Zhiquan Wen, Guanghui Xu, Mingkui Tan, Qingyao Wu, and Qi Wu. Debiased visual question answering from feature and sample perspectives. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. URL https://papers.nips.cc/paper_files/paper/2021/hash/1f4477bad7af3616c1f933a02bfabe4e-Abstract.html.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. doi: 10.1007/BF00992696.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*, 2023.

Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J. Cai. Promptchainer: Chaining large language model prompts. 2022.

Yuhuai Wu, Honghua Dong, Roger B. Grosse, and Jimmy Ba. The scattering compositional learner: Discovering objects, attributes, relationships in analogical reasoning, 2020. URL https://arxiv.org/abs/2007.04212.

Zhirong Wu, Yuanjun Xiong, Stella Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in Neural Information Processing Systems*, 28, 2015. URL https://proceedings.neurips.cc/paper/2015/hash/07563a3fe3bbe7e3ba84431ad9d055af-Abstract.html.

Jiancheng Yang, Ravi Sandhu, and Mehak Arora. Reinventing 2d convolutions for 3d images, 2019. URL https://arxiv.org/abs/1911.10477.

Lingxiao Yang, Hongzhi You, Zonglei Zhen, Dahui Wang, Xiaohong Wan, Xiaohua Xie, and Ru-Yuan Zhang. Neural prediction errors enable analogical visual reasoning in human standard intelligence tests. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 39572–39583. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/yang23r.html.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. *arXiv preprint arXiv:1703.04933*, 2017.

Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved VQGAN. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/abs/2110.04627.

John R Zech, Marcus A Badgeley, Manway Liu, Anthony B Costa, Joseph J Titano, and Eric Karl Oermann. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: a cross-sectional study. *PLoS Medicine*, 15(11):e1002683, 2018. URL https://doi.org/10.1371/journal.pmed.1002683.

Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. RAVEN: A Dataset for Relational and Analogical Visual REasoNing. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5312–5322, Long Beach, CA, USA, June 2019a. IEEE. ISBN 9781728132938. doi: 10.1109/CVPR.2019.00546. URL https://ieeexplore.ieee.org/document/8953364/.

Chi Zhang, Baoxiong Jia, Feng Gao, Yixin Zhu, HongJing Lu, and Song-Chun Zhu. Learning perceptual inference by contrasting. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019b. URL https://papers.nips.cc/paper_files/paper/2019/hash/6766aa2750c19aad2fa1b32f36ed4aee-Abstract.html.

Chi Zhang, Baoxiong Jia, Song-Chun Zhu, and Yixin Zhu. Abstract spatial-temporal reasoning via probabilistic abduction and execution. In *CVPR*, pages 9736–9746, 2021a.

Chi Zhang, Sirui Xie, Baoxiong Jia, Ying Nian Wu, Song-Chun Zhu, and Yixin Zhu. Learning algebraic representation for systematic generalization in abstract reasoning. In *ECCV*, 2022.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021b. URL https://doi.org/10.1145/3446776.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1710.09412.

Liang Zhang and Edith Aurora Graf. Mathematical computation and reasoning errors by large language models, 2025. URL https://arxiv.org/abs/2508.09932.

Yizhe Zhang, He Bai, Ruixiang Zhang, Jiatao Gu, Shuangfei Zhai, Joshua M. Susskind, and Navdeep Jaitly. How far are we from intelligent visual deductive reasoning?, 2024. URL https://arxiv.org/abs/2403.04732.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. Siren's song in the ai ocean: Hallucination in large language models, 2023. URL https://arxiv.org/abs/2309.01219.

Chengshuai Zhao, Zhen Tan, Pingchuan Ma, Dawei Li, Bohan Jiang, Yancheng Wang, Yingzhen Yang, and Huan Liu. Is chain-of-thought reasoning of LLMs a mirage? a data distribution lens, 2025. URL https://arxiv.org/abs/2508.01191.

Yupeng Zheng, Zhongpu Xia, Qichao Zhang, Teng Zhang, Ben Lu, Xiaochuang Huo, Chao Han, Yixian Li, Mengjie Yu, Bu Jin, Pengxuan Yang, Yuhang Zheng, Haifeng Yuan, Ke Jiang, Peng Jia, Xianpeng Lang, and Dongbin Zhao. Preliminary investigation into data scaling laws for imitation learning-based end-to-end autonomous driving, 2024. URL https://arxiv.org/abs/2412.02689.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2223–2232, 2017. URL https://arxiv.org/abs/1703.10593.

Tao Zhuo and Mohan S. Kankanhalli. Effective abstract reasoning with dual-contrast network. 2021. URL https://openreview.net/forum?id=ldxlzGYWDmW.

Appendix A

Appendix

A.1 Experimental Setup

This section provides the technical details behind the experiments. It covers the ACT model architecture, the hyperparameter search for the LCM, the loss-weight optimization for DCM, and a list of the code, models, and datasets released with this thesis.

A.1.1 Model Architecture

The ACT model comprises an image tokenizer, transformer, property predictor (refer to Fig. 4.1 and Sec. 4.2), and, for the LCM configurations, a comparator.

The **Tokenizer** performs the conversion of panels into token sequences using a convolutional neural network (CNN). The choice of tokenizer type (outlined in Sec. 4.2) determines whether the CNN is applied to entire RPM task images (Task tokenizer), rows of RPM task images stacked channel-wise (Row tokenizer), or to each of the nine panels independently (Panel tokenizer). The CNN is the convolutional part of EfficientNetV2B0 (Tan and Le, 2021) pre-trained on ImageNet (Deng et al., 2009) (the fully connected layers were dropped). The resulting superpixels, with 1280 channels each, undergo projection to 128 dimensions using an additional linear layer (a 1×1 convolution) to match the dimensionality of transformer's tokens.

The **Transformer** uses encoder-only transformer blocks (Vaswani et al., 2017) stacked four times, specifically following the BERT transformer block architecture (Devlin et al., 2018). Each block consists of eight independent transformer heads operating simultaneously. Within each head, the input dimensionality of the transformer is set to 128, meaning that the queries, keys, and values in the self-attention part of the head are 128-dimensional vectors. A small two-layer dense feed-forward network processes the raw tokens generated by heads, with 512 units (the *inner size* parameter) featuring the GELU activation function (Hendrycks and Gimpel, 2016) in the first layer, followed by a second linear layer comprising

128 units. The model learns positional embeddings, which are then added to the input tokens. The hyperparameters for the transformer are detailed in Table A.1.

The **Predictor** is responsible for mapping chunks of output tokens to probability distributions characterizing the properties of individual panels. Each time the predictor is invoked, it processes a single chunk of concatenated tokens, forecasting the properties of an individual panel. The predictor's input dimensionality k is contingent upon the tokenizer type and is a multiple of the transformer's output tokens dimensionality (128); more precisely 897 for the Task tokenizer (7 tokens $\times$ 128), 384 for the Row tokenizer (3 tokens $\times$ 128), and 1152 for the Panel tokenizer (9 tokens $\times$ 128). The predictor comprises two fully connected layers, each with 1000 units and GELU non-linearity, followed by an output layer containing 557 neurons that represent the combined dimensionality of the probability distributions over panel properties. The units that form the encodings of individual properties share a softmax activation function; for the present$_i$ properties that signal the presence or absence of objects in slots, a sigmoid activation function is used. Layer normalization (Ba et al., 2016) with an epsilon value of 0.001 precedes each layer.

The instances of ACT architecture were trained using the Adam optimizer employing a learning rate of 0.001 and batch size of 64.

The LCM models extend the ACT architecture by incorporating a **Comparator** comprising four hidden fully connected layers, each containing 1024 neurons using the GELU activation function. The output layer consists of a single neuron that outputs the score, representing the similarity between the two 557-dimensional input vectors of properties predicted/classified for a pair of panels. Additionally, we introduced Layer Normalization with an epsilon value of 0.001 before each layer.

To train both the ACT+LCM and ACT+CCM, we adopted the Adam optimizer with learning rate of 3.5×10^{-5} and a batch size of 32 (note that the CCM does not add any additional layers to the ACT).

The above hyperparameters for the LCM (number of hidden layers, hidden size, layer normalization placement, layer dropout placement, learning rate, batch size) were selected through a comprehensive hyperparameters search.

Table A.1: Hyperparameters of the transformer.

Hyperparameter	Value
Number of blocks	4
Number of heads	8
Size (dimensionality of tokens)	128
Inner size	512
Activation	GELU
Dropout	0.1
Layer Normalization	0.001

Table A.2: Hyperparameters for LCM and CCM training.

Parameter	Value
Learning Rate	3.5×10^{-5}
Batch Size	32
Epochs	200 (early stopping)
Optimizer	Adam

A.1.2 Implementation and Computational Resources

The software framework for conducting experiments has been written in Python ver. 3.10, with the deep learning models implemented in the TensorFlow library ver. 2.11 utilizing cuDNN ver. 8.2.1. The source code and documentation of the framework is available at `https://github.com/jakubkwiatkowski/abstract_compositional_transformer`.

Models have been trained on Tesla V100-SXM2-32GB GPU card and servers equipped with Xeon Gold 5115 processor and 16GB of RAM. Training times varied significantly by tokenizer architecture: the Row tokenizer required approximately 8 hours for random masking phase, the Task tokenizer required 11.3 hours, and the Panel tokenizer required 17.4 hours. Query masking fine-tuning required an additional

2–4 hours depending on the tokenizer (2 hours for Task, 3.2 hours for Row, and 3.9 hours for Panel). Querying a model to solve a single task on this hardware takes ∼2 seconds.

A.1.3 Tensor Shape Flow in ACT Implementation

The shape of tensors as they propagate through the ACT architecture varies depending on the tokenizer type employed. Listing A.1 presents the hierarchical data flow for three tokenizer variants (Task, Row, and Panel), illustrating how input dimensions transform at each processing stage. The implementation employs a modular structure comprising `GridTransformer` (top-level orchestrator), `ImageTransformer` (image processing pipeline), and nested components that correspond to the three conceptual parts of ACT described in Section 4.2: the **Image Tokenizer** (encompassing `Preprocessor`, `Tokenizer`, and `Extractor` with `Encoder` and `Pooling`), the **Transformer** (processing token sequences via multi-head attention), and the **Predictor** (implemented within `Detokenizer`, mapping outputs to 557-dimensional property distributions).

Task Tokenizer Pipeline:

```
GridTransformer in: {'inputs': [16, 160, 160], 'target': [16, 113], 'index': 1}
    ImageTransformer in: [[9, 160, 160], 9]
        Preprocessor in: [[9, 160, 160], 9]
            Tokenizer in: [84, 84, 9]
                Extractor in: [252, 252, 3]
                    Encoder in: [252, 252, 3]
                    Encoder out: [8, 8, 1280]
                    Pooling in: [8, 8, 128]
                    Pooling out: [64, 128]
                Extractor out: [64, 128]
            Tokenizer out: [64, 128]
        Preprocessor out: [64, 128]
        Transformer in: [64, 128]
            Detokenizer in: [64, 128]
            Detokenizer out: [9, 896]
        Transformer out: 557
    ImageTransformer out: 557
GridTransformer out: {'inputs': [9, 160, 160], 'target': [9, 113], 'index': 1,
                      'labels': [16, 113], 'mask': 9, 'output': 557}
```

Row Tokenizer Pipeline:

```
GridTransformer in: {'inputs': [16, 160, 160], 'target': [16, 113], 'index': 1}
    ImageTransformer in: [[9, 160, 160], 9]
        Preprocessor in: [[9, 160, 160], 9]
            Tokenizer in: [84, 84, 9]
                Extractor in: [3, 84, 84, 3]
                    Encoder in: [3, 84, 84, 3]
                    Encoder out: [3, 3, 3, 1280]
                    Pooling in: [3, 3, 3, 128]
                    Pooling out: [27, 128]
```

```
                Extractor out: [27, 128]
            Tokenizer out: [27, 128]
        Preprocessor out: [27, 128]
        Transformer in: [27, 128]
            Detokenizer in: [27, 128]
            Detokenizer out: [9, 384]
        Transformer out: 557
    ImageTransformer out: 557
GridTransformer out: {'inputs': [9, 160, 160], 'target': [9, 113], 'index': 1,
                      'labels': [16, 113], 'mask': 9, 'output': 557}
```

Panel Tokenizer Pipeline:

```
GridTransformer in: {'inputs': [16, 160, 160], 'target': [16, 113], 'index': 1}
    ImageTransformer in: [[9, 160, 160], 9]
        Preprocessor in: [[9, 160, 160], 9]
            Tokenizer in: [84, 84, 9]
                Extractor in: [9, 84, 84, 3]
                    Encoder in: [9, 84, 84, 3]
                    Encoder out: [9, 3, 3, 1280]
                    Pooling in: [9, 3, 3, 128]
                    Pooling out: [81, 128]
                Extractor out: [81, 128]
            Tokenizer out: [81, 128]
        Preprocessor out: [81, 128]
        Transformer in: [81, 128]
            Detokenizer in: [81, 128]
            Detokenizer out: [9, 1152]
        Transformer out: [9, 557]
    ImageTransformer out: [9, 557]
GridTransformer out: {'inputs': [9, 160, 160], 'target': [9, 113], 'index': 1,
                      'labels': [16, 113], 'mask': 9, 'output': [9, 557]}
```

Listing A.1: Complete hierarchical tensor shape transformations in ACT implementation across three tokenizer variants. The Task tokenizer processes the entire RPM puzzle (252×252 pixels) producing 64 tokens; the Row tokenizer processes three rows independently (stacked channel-wise) producing 27 tokens; the Panel tokenizer processes each of 9 panels individually producing 81 tokens. Each tokenizer variant demonstrates distinct tensor flow patterns through the Encoder (EfficientNetV2B0 producing 1280-channel features), Pooling (reducing to 128-dimensional tokens), Transformer (processing token sequences), and Detokenizer/Predictor (generating 557-dimensional property distributions).

A.1.4 Loss Weight Hyperparameter Optimization

In training, contributions of individual properties are weighed in the loss function using weights determined empirically at the beginning of the project. These are:

- 1.000 for *arrangement*,
- 2.834 for $present_i$,

- 0.852 for *color*,
- 1.096 for *size*,
- 1.219 for *shape*.

As described in Sec. 4.3.3, DCM employs weighted property-wise losses to account for the varying importance and difficulty of different properties in the prediction task. Initially, we established baseline weights through preliminary experiments by computing the relative loss contributions of each property across the dataset and using these contributions to balance the overall loss function. These baseline weights provided good performance across all benchmarks. However, to assess the potential for further improvement, we conducted systematic hyperparameter optimization to discover dataset-specific optimal weight configurations.

Optimization methodology. We performed independent hyperparameter optimization for each of the three benchmarks (RAVEN, I-RAVEN, and RAVEN-FAIR), searching for weight configurations that maximized validation performance on each dataset. The optimization was carried out using Tree-structured Parzen Estimation (TPE) (Bergstra et al., 2011). The optimization process explored the five-dimensional weight space corresponding to the five property categories used in the loss function. To evaluate the generalization properties of these optimized weights, we conducted cross-dataset evaluation: each optimized weight configuration was tested on all three benchmarks, allowing us to assess whether dataset-specific optimization produces overfitted weights or transferable improvements.

Results. Table A.3 presents the AccProb metric across all combinations of optimization datasets (columns) and evaluation datasets (rows). The "Base" column shows performance using the original baseline weights, while the remaining columns show performance when using weights optimized for each specific dataset.

Table A.3: Cross-dataset evaluation of optimized loss weights. Columns indicate which dataset was used for optimization; rows indicate evaluation dataset. Values represent AccProb (%).

Evaluation	Base	RAVEN	I-RAVEN	RAVEN-FAIR
RAVEN	96.97	97.07	96.77	97.05
I-RAVEN	95.39	95.68	95.80	95.64
RAVEN-FAIR	98.05	98.50	98.29	98.54

The optimized weight configurations discovered through the hyperparameter search are presented in Table A.4 alongside the baseline weights for direct comparison. Each configuration consists of five weight values corresponding to the five property categories in DCM's loss function.

Table A.4: Baseline and optimized loss weight configurations for DCM. Rows list the weights used in each configuration; columns correspond to the five property categories in DCM's loss function. The baseline weights are identical across datasets by construction; the optimized weights are the result of the hyperparameter search described above.

Configuration	Arrangement	present$_i$	Color	Size	Type
Baseline	1.000	2.834	0.852	1.096	1.219
RAVEN	1.000	0.851	1.949	2.721	1.862
I-RAVEN	1.000	1.503	0.759	0.792	0.881
RAVEN-FAIR	1.000	3.820	2.236	3.309	1.982

These hyperoptimized weights were *not* used in the experimental results reported throughout this thesis. This optimization study was conducted as an exploratory analysis to assess the potential ceiling for DCM performance improvement through loss reweighting. The results reported in the main chapters use the baseline weights established through preliminary experiments, ensuring fair comparison across all experimental configurations.

A.1.5 Resources

All code, trained models, and datasets accompanying this thesis are released publicly. Each resource below is paired with a short description of what it contains.

Source Code

- **Abstract Compositional Transformer** — `https://github.com/jakubkwiatkowski/abstract_compositional_transformer`
  Primary repository with the ACT architecture, training pipeline, choice makers (DCM, LCM, CCM), evaluation scripts, and visualizations.
- **core_tools** — `https://github.com/jakubkwiatkowski/core_tools`
  General-purpose utilities for deep learning experiments (configuration, logging, checkpointing, distributed training).
- **compositional_transformer** — `https://github.com/jakubkwiatkowski/compositional_transformer`
  Generic implementation of grid-based transformer architectures with flexible tokenization, positional encodings, and predictor modules.
- **raven_tools** — `https://github.com/jakubkwiatkowski/raven_tools`
  RPM-specific utilities: property-vector encoding, panel rendering, dataset loading, and evaluation metrics for the RAVEN family benchmarks.

Trained Models and Demonstrations

- **Model weights (Google Drive)** — `https://drive.google.com/drive/folders/1kxfS_QATWctTJFV3lzM99U83CnKV1zFU`
  Pretrained weights for all ACT and choice-maker configurations evaluated in the thesis.
- **Hugging Face model** (`jkwiatkowski/raven`) — `https://huggingface.co/jkwiatkowski/raven`
  Best-performing ACT property prediction model (Task tokenizer, Combined masking) ready for direct use in downstream applications.
- **Hugging Face Space** (`jkwiatkowski/raven`) — `https://huggingface.co/spaces/jkwiatkowski/raven`
  Interactive web demo for visualizing ACT predictions on RAVEN tasks.

Datasets

- **Original RAVEN** — `https://github.com/WellyZhang/RAVEN`
  The original RAVEN benchmark (Zhang et al., 2019a) with 70,000 training tasks and 10,000 test tasks across 7 configurations.
- **Hugging Face RAVEN dataset** (`jkwiatkowski/raven`) — `https://huggingface.co/datasets/jkwiatkowski/raven`
  RAVEN reformatted for the Hugging Face Datasets library with built-in splits and caching.
- **RAVEN Properties dataset** (`jkwiatkowski/raven_properties`) — `https://huggingface.co/datasets/jkwiatkowski/raven_properties`

RAVEN extended with ground-truth property vectors for all panels, enabling direct evaluation of property prediction accuracy.

A.2 Visualizations

A.2.1 DCM Visualizations

Figures A.1 and A.2 present the visualizations of the behaviors of the Row and Task models on 8 additional tasks, presented in the same way as the visualizations described in Sec. A.2.1. All models were trained in Combined masking mode. Additional visualizations are available in the project repository.

The layout of visualization in each inset is as follows. **Left:** the task. **Middle:** the correct answer and the rendering of models' predictions p (Step 1 of DCM) for the Row and Task model. **Right:** the answer panels and the renderings of classifications p_i generated by the Row and Task model (Step 2 of the DCM). The panels corresponding to the most similar property vectors are marked with thicker borders. Predictions and classifications are rendered from property vectors produced by the model while fixing rotation angles, as the angle is irrelevant in RAVEN tasks. To facilitate analysis, we render the *color* property using pseudocoloring (it is conventionally rendered in grayscale).

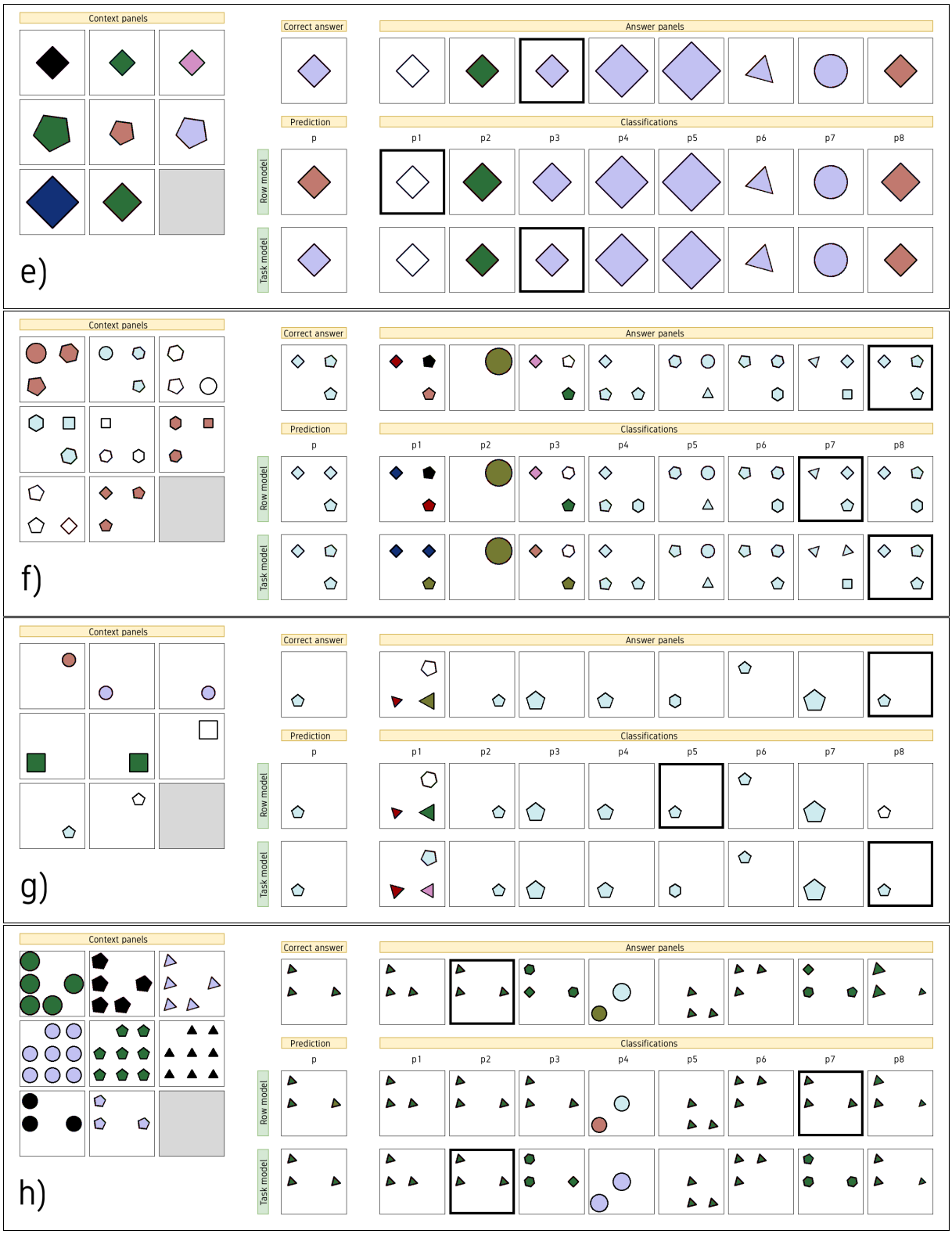


FIGURE A.1: Row and Task model comparison on additional tasks (E–H) from the RAVEN benchmark (Combined masking mode). Left: task; middle: correct answer with Row and Task model predictions p; right: answer panels with classifications p_i, most similar marked with thicker border.

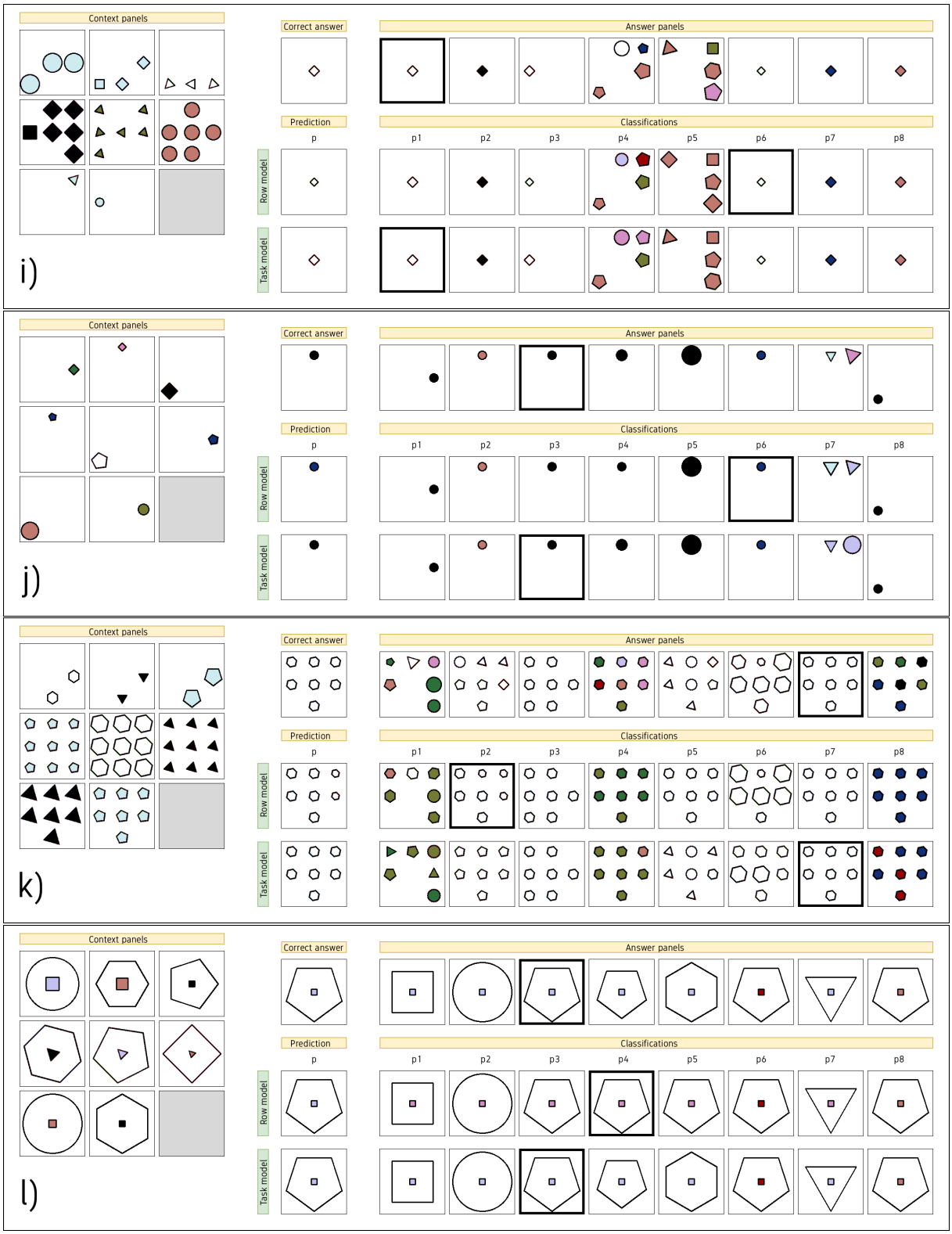


FIGURE A.2: Row and Task model comparison on additional tasks (I–L) from the RAVEN benchmark (Combined masking mode). Left: task; middle: correct answer with Row and Task model predictions p; right: answer panels with classifications p_i, most similar marked with thicker border.

A.2.2 DCM vs LCM Visualizations

Figures A.3 and A.4 present the visualizations comparing the behaviors of the DCM and LCM choice makers on 12 additional tasks from the RAVEN benchmark. Both models were trained using the Com-

bined masking mode with ACT as the property predictor. Additional visualizations are available in the project repository.

The layout of visualization in each inset is as follows. **Left:** the task. **Middle:** the correct answer and the rendering of the ACT model's prediction p for the missing panel's properties. **Right:** the answer panels and the renderings of the selected answers by DCM and LCM, respectively. The panels corresponding to the choices made by each method are marked with thicker borders. Predictions are rendered from property vectors produced by the model while fixing rotation angles, as the angle is irrelevant in RAVEN tasks. To facilitate analysis, we render the *color* property using pseudocoloring (it is conventionally rendered in grayscale). These visualizations illustrate the differences in reasoning strategies between the distance-based approach of DCM and the learned classification approach of LCM.

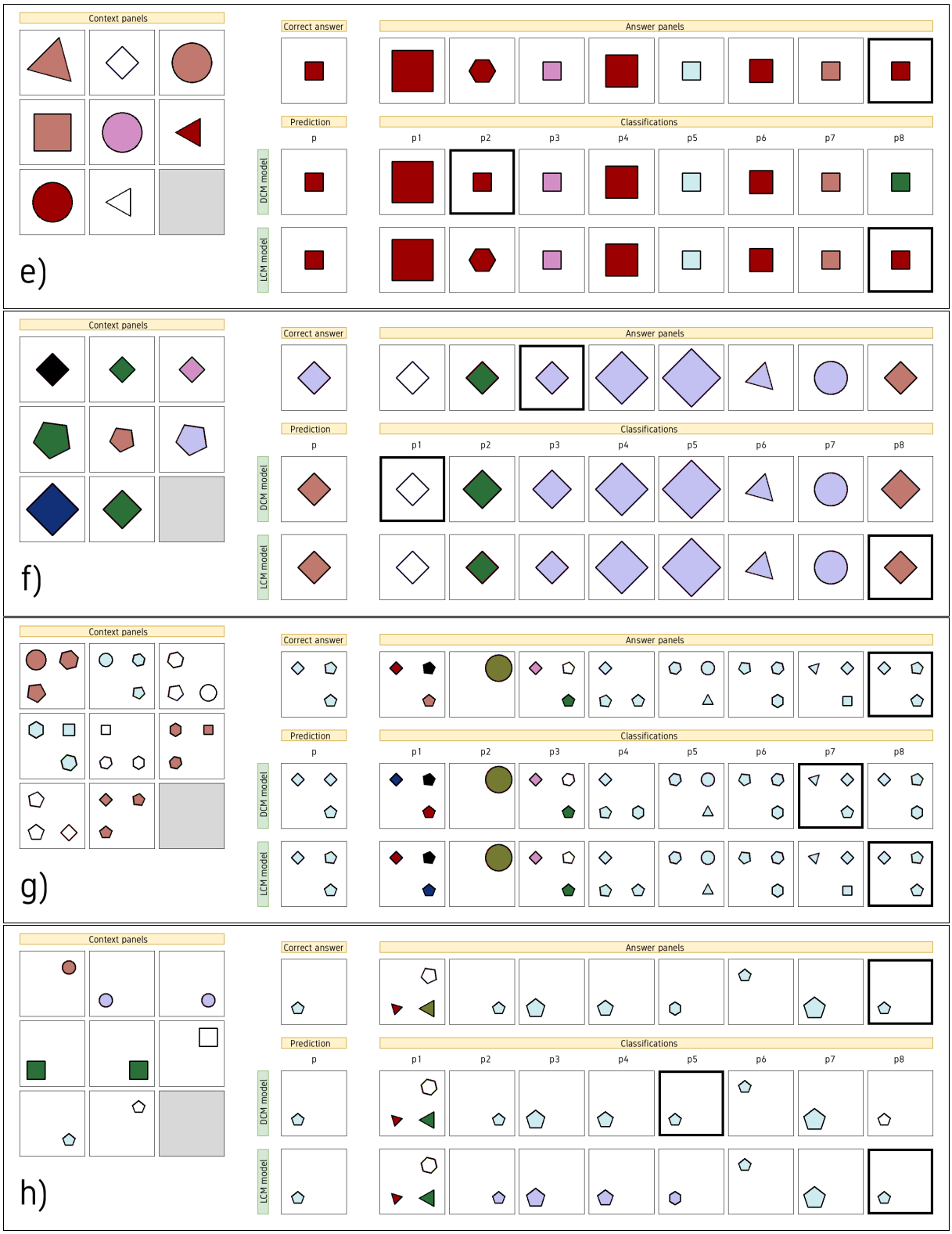


FIGURE A.3: DCM vs. LCM choice on additional tasks (E–H) from the RAVEN benchmark (Combined masking mode). Left: task; middle: correct answer with ACT prediction p; right: answer panels with DCM and LCM selections marked with thicker border.

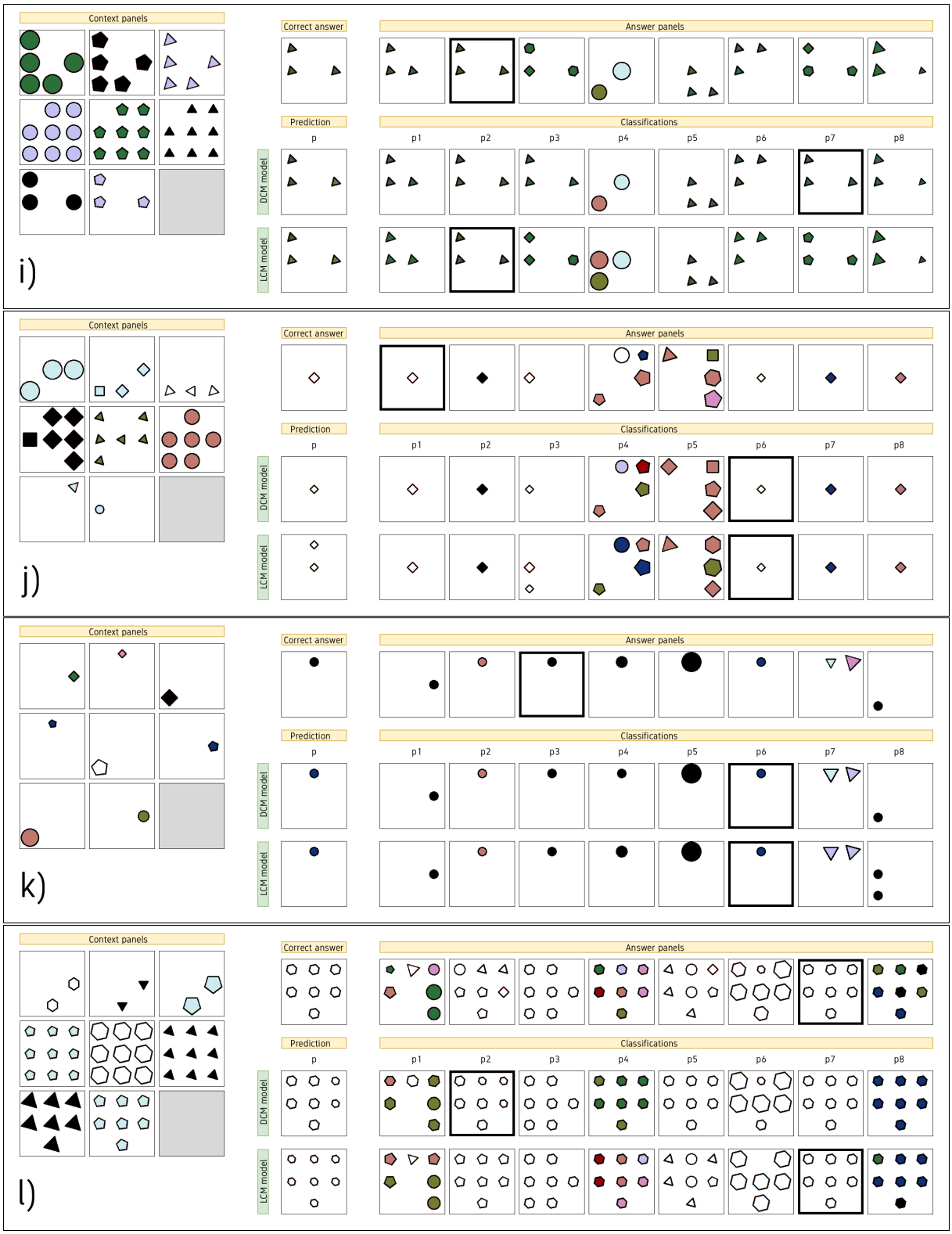


FIGURE A.4: DCM vs. LCM choice on additional tasks (I–L) from the RAVEN benchmark (Combined masking mode). Left: task; middle: correct answer with ACT prediction p; right: answer panels with DCM and LCM selections marked with thicker border.

A.2.3 Additional PGM Visualizations

Figures A.5 and A.6 present additional visualizations of DCM behavior on PGM tasks, following the same format as Figs. 6.1 and 6.2. These examples further illustrate the model's generalization patterns,

reconstruction strategies, and limitations when confronting out-of-distribution visual elements from the PGM dataset while trained exclusively on RAVEN. Additional visualizations are available in the project repository.

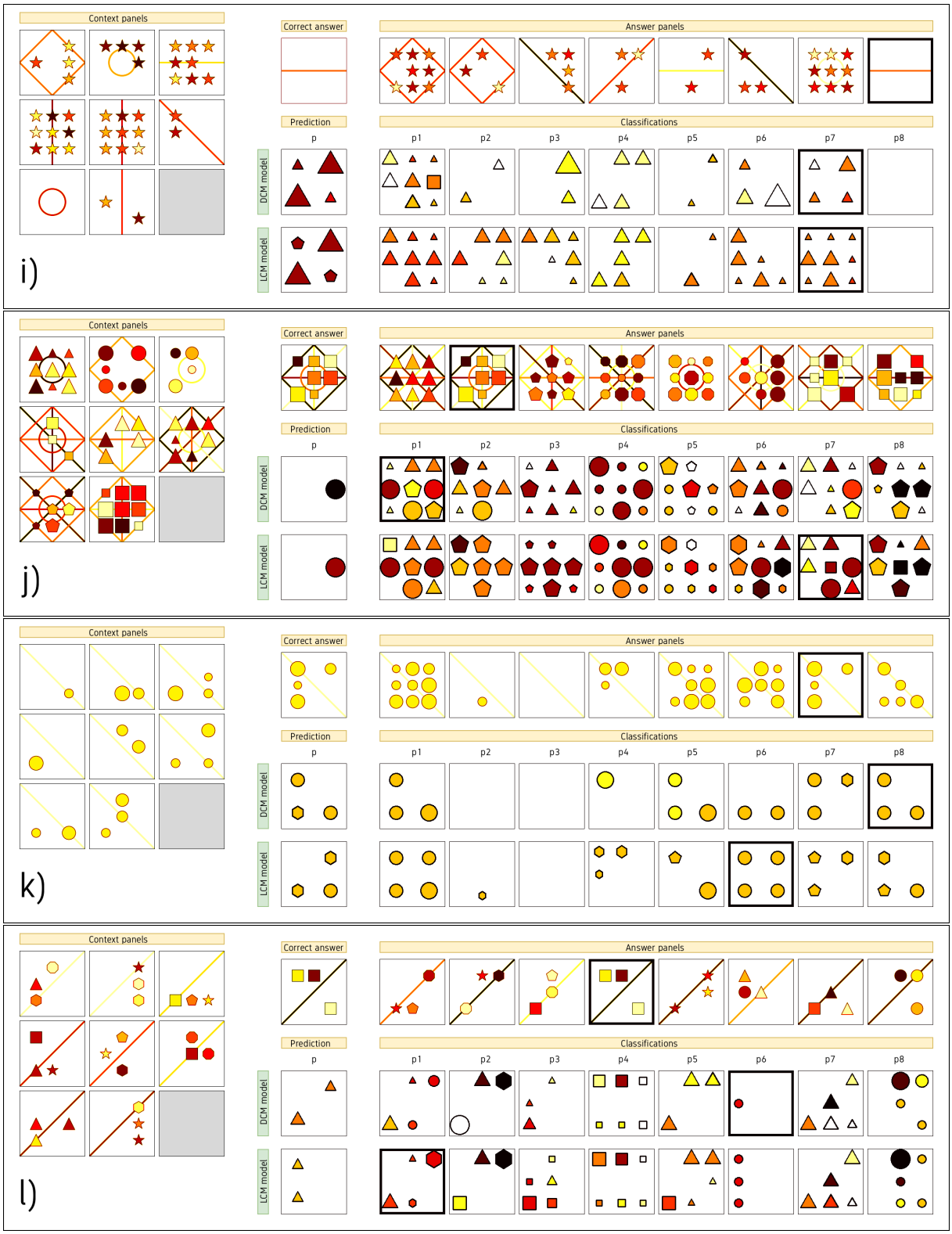


FIGURE A.5: Additional PGM generalization examples (Part 1, tasks I–L). DCM trained on RAVEN reconstructs PGM panels using learned RAVEN representations. Left: PGM input; middle: correct answer with predicted panel reconstruction; right: reconstructions of all answer options with model selection indicated by thicker border.

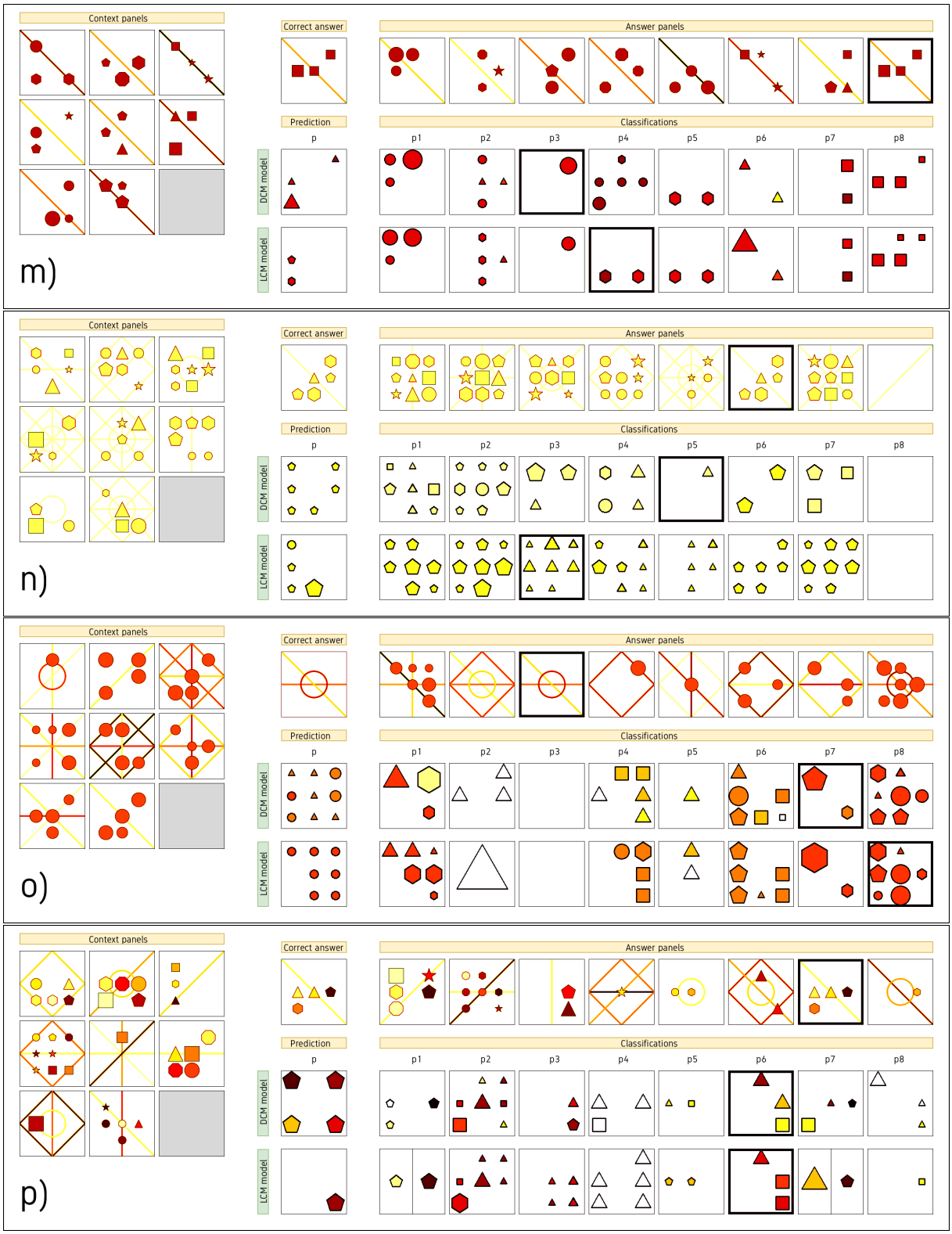


FIGURE A.6: Additional PGM generalization examples (Part 2, tasks M–P). DCM trained on RAVEN reconstructs PGM panels using learned RAVEN representations. Left: PGM input; middle: correct answer with predicted panel reconstruction; right: reconstructions of all answer options with model selection indicated by thicker border.

A.3 Detailed Similarity Bias Analysis

This appendix provides detailed experiments supporting the similarity bias discussion in Section 3.2. We evaluate similarity-based solvers across multiple *representation levels* and multiple *context aggrega-*

tion strategies, to quantify how easily visual similarity can be exploited in **RAVEN**, **I-RAVEN**, and **RAVEN-FAIR**.

A.3.1 Image-Based Similarity Heuristics

We first evaluate direct pixel-level similarity using standard image quality metrics (defined in Table A.5). The five context aggregation strategies (C6, C8, C6+C8, Heuristic, Mean) are the same as those described in Sec. 3.2.1; the reader is referred there for full definitions. Tables A.5–A.7 report accuracy of each metric–strategy combination on the RAVEN, I-RAVEN, and RAVEN-FAIR test sets, respectively. Each cell shows the percentage of correct answer selections made by a similarity-based solver that scores candidates using the given metric and context strategy.

Table A.5: Accuracy (%) of image similarity heuristics on **RAVEN** test set. MSE = Mean Squared Error, RMSE = Root Mean Squared Error, PSNR = Peak Signal-to-Noise Ratio, UQI = Universal Quality Index (Wang and Bovik, 2002), ERGAS = Erreur Relative Globale Adimensionnelle de Synthèse (Wald, 2000), RASE = Relative Average Spectral Error (Alparone et al., 2004), SAM = Spectral Angle Mapper (Kruse et al., 1993), PSNR-B = PSNR with Blocking Effect Factor.

Metric	C6	C8	C6+C8	Heuristic	Mean
MSE	6.22	8.04	7.69	8.76	6.44
RMSE	6.22	8.04	8.19	8.76	6.74
PSNR	6.22	8.04	8.13	8.76	7.27
UQI	6.06	8.14	7.64	8.47	6.30
ERGAS	6.22	8.04	8.17	8.64	6.86
RASE	6.98	8.54	9.34	9.31	9.14
SAM	6.09	7.92	7.97	8.63	6.44
PSNR-B	6.22	8.06	8.14	8.79	7.25

Table A.6: Accuracy (%) of image similarity heuristics on **I-RAVEN** test set.

Metric	C6	C8	C6+C8	Heuristic	Mean
MSE	15.50	17.28	17.43	18.69	14.85
RMSE	15.50	17.28	17.85	18.69	15.27
PSNR	15.50	17.28	17.64	18.69	15.64
UQI	15.00	16.96	16.83	17.55	14.72
ERGAS	15.50	17.28	17.88	18.54	15.41
RASE	14.78	17.52	17.16	18.13	14.76
SAM	15.25	17.16	17.56	18.39	15.29
PSNR-B	15.49	17.29	17.63	18.69	15.64

Table A.7: Accuracy (%) of image similarity heuristics on **RAVEN-FAIR** test set.

Metric	C6	C8	C6+C8	Heuristic	Mean
MSE	15.29	17.21	17.57	18.82	14.93
RMSE	15.29	17.21	18.03	18.82	15.41
PSNR	15.29	17.21	17.89	18.83	16.21
UQI	14.56	17.56	17.06	18.12	15.04
ERGAS	15.29	17.21	18.14	18.76	15.60
RASE	14.44	17.79	17.47	18.43	15.51
SAM	15.21	17.17	17.70	18.69	15.34
PSNR-B	15.31	17.23	17.90	18.86	16.20

A.3.2 Learned Feature Similarity

We next evaluate the same context aggregation strategies using deep feature embeddings. We consider two feature sources: (i) ImageNet-pretrained EfficientNetV2 features used as a frozen general-purpose perceptual embedding (no RAVEN adaptation), and (ii) the same backbone adapted by training on RAVEN property prediction (RAVEN-adapted features).

ImageNet-Pretrained EfficientNetV2 Features (Frozen; No RAVEN Adaptation)

Tables A.8–A.10 report performance using frozen ImageNet-pretrained EfficientNetV2 embeddings.

Table A.8: Accuracy (%) using ImageNet-pretrained EfficientNetV2 features (frozen; no RAVEN adaptation) on **RAVEN**.

Metric	C6	C8	C6+C8	Heuristic	Mean
Cosine	6.24	7.16	8.05	8.54	5.68
MSE	6.61	7.34	8.16	8.66	6.21
MAE	6.69	7.32	8.46	8.81	7.09

Table A.9: Accuracy (%) using ImageNet-pretrained EfficientNetV2 features (frozen; no RAVEN adaptation) on **I-RAVEN**.

Metric	C6	C8	C6+C8	Heuristic	Mean
Cosine	16.99	17.67	18.86	19.59	15.25
MSE	16.86	17.51	18.56	19.46	15.09
MAE	16.61	17.06	18.76	19.18	15.57

Table A.10: Accuracy (%) using ImageNet-pretrained EfficientNetV2 features (frozen; no RAVEN adaptation) on **RAVEN-FAIR**.

Metric	C6	C8	C6+C8	Heuristic	Mean
Cosine	16.85	17.18	18.75	20.16	13.64
MSE	16.94	17.48	19.22	20.36	13.81
MAE	16.88	17.42	19.68	20.32	15.14

RAVEN-Adapted EfficientNetV2 Features (Property-Prediction Trained)

Tables A.11–A.13 report performance using the same EfficientNetV2 backbone after adaptation through RAVEN property prediction training.

Table A.11: Accuracy (%) using RAVEN-adapted EfficientNetV2 features on **RAVEN**.

Metric	C6	C8	C6+C8	Heuristic	Mean
Cosine	5.46	6.11	6.60	7.52	5.17
MSE	4.94	5.86	6.51	7.21	6.34
MAE	5.23	6.19	7.59	7.64	7.60

Table A.12: Accuracy (%) using RAVEN-adapted EfficientNetV2 features on **I-RAVEN**.

Metric	C6	C8	C6+C8	Heuristic	Mean
Cosine	15.12	17.82	19.74	20.71	19.81
MSE	14.96	17.79	17.94	19.71	17.58
MAE	15.19	17.99	19.15	20.06	19.39

Table A.13: Accuracy (%) using RAVEN-adapted EfficientNetV2 features on **RAVEN-FAIR**.

Metric	C6	C8	C6+C8	Heuristic	Mean
Cosine	12.41	15.78	16.55	19.04	15.84
MSE	13.51	16.29	16.47	18.55	15.99
MAE	13.61	16.96	18.06	19.26	18.34

A.3.3 Structural Distance Analysis

To quantify similarity bias at the property level, we analyze Hamming distances and absolute property differences between context panels and answer panels in the ground-truth property space, following the notation introduced in Sec. 3.2.1. The 3×3 grids represent context panel positions, with position (2,2) being the query panel (distance 0 by definition).

Hamming Distance Analysis

Table A.14 presents mean Hamming distances between each context panel position and answer panels (correct vs. incorrect) across all three benchmarks, extending the analysis introduced in Sec. 3.2.1.

Table A.14: Mean Hamming distance between context panels and answer panels across RAVEN benchmarks. Panel C8 (row 2, column 1) consistently shows minimal distance to correct answers. Discrimination margins (difference between C8 distances to correct vs. incorrect answers) are 6.37 (RAVEN), 10.07 (I-RAVEN), and 11.24 (RAVEN-FAIR) Hamming units.

Correct Answer

	R\C	0	1	2
RAVEN	0	78.62	75.90	80.33
	1	78.10	76.63	79.92
	2	68.93	**68.70**	0.00

	R\C	0	1	2
I-RAVEN	0	78.22	76.28	80.58
	1	77.82	76.55	79.27
	2	70.05	**69.73**	0.00

	R\C	0	1	2
RAVEN-FAIR	0	77.98	76.77	80.47
	1	78.87	76.28	80.58
	2	69.73	**69.62**	0.00

Incorrect Answers (Average)

	R\C	0	1	2
RAVEN	0	80.77	79.22	82.06
	1	80.48	79.58	81.75
	2	75.11	**75.07**	0.00

	R\C	0	1	2
I-RAVEN	0	82.31	82.12	83.55
	1	82.45	81.82	83.00
	2	80.07	**79.80**	0.00

	R\C	0	1	2
RAVEN-FAIR	0	82.71	82.26	83.33
	1	82.55	81.68	83.23
	2	80.67	**80.86**	0.00

Absolute Property Difference Analysis

Table A.15 presents analogous analysis using mean absolute difference in property values (continuous metric) rather than Hamming distance, following the notation of Sec. 3.2.1. This metric captures the magnitude of property differences rather than binary mismatch counts.

Table A.15: Mean absolute property difference between context panels and answer panels across RAVEN benchmarks. Panel C8 shows minimal difference to correct answers: 1.56 (RAVEN), 1.60 (I-RAVEN), 1.61 (RAVEN-FAIR). Discrimination margins are 0.26 (RAVEN), 0.42 (I-RAVEN), and 0.43 (RAVEN-FAIR) property values.

Correct Answer

	R\C	0	1	2
RAVEN	0	2.35	2.00	1.95
	1	2.35	2.01	1.98
	2	1.86	**1.56**	0.00

	R\C	0	1	2
I-RAVEN	0	2.33	2.01	1.98
	1	2.35	2.02	1.96
	2	1.90	**1.60**	0.00

	R\C	0	1	2
RAVEN-FAIR	0	2.33	2.04	1.97
	1	2.37	2.01	1.96
	2	1.90	**1.61**	0.00

Incorrect Answers (Average)

	R\C	0	1	2
RAVEN	0	2.31	2.07	2.09
	1	2.32	2.08	2.11
	2	2.02	**1.82**	0.00

	R\C	0	1	2
I-RAVEN	0	2.29	2.14	2.22
	1	2.31	2.15	2.21
	2	2.14	**2.02**	0.00

	R\C	0	1	2
RAVEN-FAIR	0	2.30	2.16	2.20
	1	2.30	2.13	2.20
	2	2.15	**2.04**	0.00

The consistent pattern across Hamming distance and absolute difference metrics confirms that similarity bias is structural and inherent to RPM task design, requiring careful answer set construction to mitigate exploitation.

A.4 Sudoku: Framework Validation

To validate the generalizability of our framework beyond RPM tasks, we evaluated the ACT architecture on Sudoku puzzles—a domain structurally similar to RPM but with different reasoning requirements. This experiment demonstrates that our modular architecture and self-supervised training methodology extend to related abstract reasoning tasks without significant architectural modifications.

A.4.1 Task Formulation and Similarity to RPM

Sudoku presents a grid-based reasoning task with structural parallels to RPM. The input consists of a 9×9 grid where cells contain digits (1–9) or are empty, and the objective is to predict values for empty cells such that each row, column, and 3×3 sub-grid contains each digit exactly once. This formulation mirrors RPM in several key aspects:

- **Grid structure**: Both tasks involve reasoning over 2D spatial arrangements
- **Masked prediction**: Like RPM's masked panel prediction, Sudoku requires inferring missing values from visible context
- **Constraint satisfaction**: Both require identifying and applying rules that govern relationships between grid positions
- **Compositional reasoning**: Solutions emerge from combining local constraints (sub-grids, rows, columns) with global consistency

The key difference lies in the nature of rules: RPM involves discovering implicit visual patterns from examples, while Sudoku enforces explicit, fixed logical constraints. Despite Sudoku's deterministic nature—with only a handful of simple, well-defined rules—we hypothesized that a sufficiently capable

model would learn these rigid rules symbolically. However, our findings reveal that even on such structurally simple tasks, deep learning models predominantly rely on statistical pattern recognition rather than abstract rule acquisition.

A.4.2 Model Adaptation and Training

Adapting our ACT framework to Sudoku required minimal architectural changes, validating the modularity and generalizability of our design. We evaluated two input representations to assess how the tokenization strategy affects performance:

- **Image-based representation**: Following the RPM paradigm, each Sudoku puzzle was rendered as a grid of images where individual cells contained MNIST digit images or were empty. This representation utilized the CNN-based tokenizer from the original ACT architecture, maintaining consistency with the RPM solving approach.
- **Digit-based (embedding) representation**: We replaced the CNN tokenizer with a simple embedding layer that maps digit values (0–9, where 0 represents empty cells) to 128-dimensional vectors. This more direct representation eliminates the visual encoding step and operates on symbolic digit identities.

The four-layer, 8-head BERT-style transformer was used without modification across both representations. Positional encodings capture the 2D grid structure through learned embeddings for each of the 81 cell positions. The property predictor was replaced with a 10-way classifier (digits 1–9 plus empty) operating on transformer output tokens. We employed the same self-supervised masking strategy used for RPM: during training, a random subset of filled cells was masked, and the model learned to predict their values from visible context.

The ease of adaptation—requiring changes only to input/output layers while preserving the core transformer architecture—demonstrates that our framework's design principles (modular components, self-supervised learning, flexible positional encoding) generalize beyond RPM-specific requirements.

A.4.3 Experimental Setup and Evaluation Metrics

Following the RPM evaluation paradigm, our Sudoku model produces predictions for all 81 cells in the grid, including both cells that are visible in the input puzzle (classification task) and cells that are empty and must be inferred (prediction task). This dual capability allows us to assess both the model's ability to correctly recognize given information and its capacity to reason about missing values.

We evaluated ACT performance using four complementary metrics that capture different aspects of Sudoku solving capability:

- **Acc (Accuracy)**: Mean per-cell accuracy computed only on digits that are not present in the input (the originally empty cells the model must predict). For each puzzle, this is the fraction of correctly predicted digits among all empty positions, averaged over the dataset. This is the primary metric for evaluating prediction capability.
- **Correct**: Strict puzzle-level accuracy computed only on originally empty cells. For a given puzzle, this metric equals 1 if all to-be-predicted digits are correct, and 0 otherwise, then averaged over puzzles. This represents the percentage of puzzles solved perfectly and is the most stringent evaluation of prediction performance.

- **FullAcc (Full Accuracy)**: Mean per-cell accuracy over all 81 cells, including both originally visible and originally empty positions. This metric evaluates the model's joint classification and prediction capability, assessing whether it can simultaneously recognize given digits and infer missing ones.
- **FullCorrect (Full Correct)**: Strict puzzle-level accuracy over all 81 cells. For a given puzzle, this metric equals 1 if every digit (visible and non-visible) is predicted correctly, and 0 otherwise. Even a single misclassification of a visible digit or misprediction of an empty cell results in 0 for that puzzle. This is the most stringent overall evaluation metric.

The distinction between Acc/Correct (prediction-only) and FullAcc/FullCorrect (prediction and classification) is crucial: the former metrics assess only the reasoning capability required to infer missing values, while the latter additionally penalize failures to correctly recognize already-visible information. Similarly, the distinction between mean metrics (Acc, FullAcc) and strict metrics (Correct, FullCorrect) reveals how errors accumulate across the many predictions required to complete a puzzle.

A.4.4 Results

Tables A.16 and A.17 report the results.

Table A.16: ACT performance on Sudoku with different input representations in standard (single-step) evaluation. Acc: mean per-cell accuracy on empty cells. Correct: fraction of puzzles with all empty cells predicted correctly. FullAcc: mean per-cell accuracy on all 81 cells. FullCorrect: fraction of puzzles with all 81 cells correct.

Representation	Split	Acc (%)	Correct (%)	FullAcc (%)	FullCorrect (%)
Image-based	Validation	99.52	92.40	99.66	91.71
	Test	99.82	96.02	99.87	91.35
Digit-based (embedding)	Validation	99.88	96.65	99.93	96.65
	Test	99.87	96.39	99.92	96.39

Table A.17: Standard vs. autoregressive evaluation on Sudoku. Autoregressive mode iteratively fills cells, conditioning each prediction on previously predicted values. Correct: fraction of puzzles with all empty cells predicted correctly.

Representation	Mode	Validation Correct (%)	Test Correct (%)
Image-based	Standard	92.40	96.02
	Autoregressive	97.60	99.20
Digit-based (embedding)	Standard	96.65	96.39
	Autoregressive	100.0	100.0

Both representations achieved strong in-distribution performance. The digit-based (embedding) model reached 99.87% cell-level accuracy and 96.39% strict puzzle accuracy on the test set, while the image-based model achieved 99.82% and 96.02% respectively. In autoregressive mode, the digit-based (embedding) model achieved perfect 100% correct puzzle solutions on both validation and test sets, demonstrating strong internal consistency when predictions are conditioned sequentially on previously filled cells.

We also evaluated generalization to out-of-distribution masking patterns, where the number and spatial distribution of masked cells differed from the training data. The models exhibited a severe performance drop in this setting: for the image-based model, strict puzzle accuracy (Correct), which reached 96.02% in-distribution, fell to approximately 18–23% on the OOD test set. This indicates that the models overfit to the specific masking distribution of the training set and were unable to generalize to puzzles with a different number or arrangement of masked cells—despite the underlying Sudoku rules remaining identical. Developing models that are robust to varying masking distributions is left as an important direction for future work. A comprehensive visual and explainability analysis, including attention pattern visualizations, is available in the project repository for interested readers.

A.5 LLM on RAVEN

This appendix documents the experiment introduced in Sec. 6.4 and detailed in Sec. 6.4.2, in which three model sizes from the Qwen3.5 family (0.8B, 4B, and 27B) are evaluated on a 700-item subset of **I-RAVEN** (Sec. 3.2.3) under the eight prompting strategies (S1–S8) described in Sec. 6.4.2 and summarized in Table 6.1. The experiment is designed to test the working memory hypothesis (Sec. 6.4.1): that intermediate reasoning generation—rather than model size, prompt engineering, or in-context examples—is the leading driver of accuracy in abstract reasoning tasks. Overall accuracy by model size and strategy is reported in Table 6.2; the present appendix provides the supporting per-arrangement breakdown in Sec. A.5.1 and one example prompt template in Sec. A.5.2.

A.5.1 Per-Arrangement Results

The table below reports the full per-arrangement accuracy breakdown for all three model sizes and all eight prompting strategies, complementing the overall-accuracy summary in Table 6.2. The seven spatial arrangement types (CS, D4, D9, ICO, IDO, LR, UD) are defined in Sec. 3.1, and the I-RAVEN benchmark on which the evaluation set is built is described in Sec. 3.2.3. Reading the table by column shows how each strategy performs across the seven arrangements within a single model size; reading it by row shows how each model size handles a single arrangement across strategies. Together with the working-memory analysis in Sec. 6.4.4, this breakdown supports the conclusion that structured chain-of-thought (S3–S6) and prompt chaining (S7–S8) yield large gains over the answer-only baselines (S1–S2) precisely on the arrangements that place the greatest demand on the model's working memory.

Table A.18: Per-arrangement accuracy (%) for all models and strategies. CS = center-single; D4 = distribute-four; D9 = distribute-nine; ICO = in-center-single-out-center-single; IDO = in-distribute-four-out-center-single; LR = left-center-single-right-center-single; UD = up-center-single-down-center-single.

Model	Strategy	Overall	CS	D4	D9	ICO	IDO	LR	UD
0.8B	S1	13.9	17	15	15	10	15	10	15
	S2	13.7	18	17	15	3	13	15	15
	S3	12.7	19	11	12	13	9	14	11
	S4	11.9	15	8	11	12	13	16	8
	S5	13.6	12	15	10	22	13	10	13
	S6	12.6	15	14	13	9	16	7	14
	S7	12.1	11	16	11	15	7	14	11
	S8	15.1	20	24	15	12	9	8	18
4B	S1	15.0	17	18	10	13	16	17	14
	S2	14.1	20	15	15	9	11	15	14
	S3	63.4	75	56	51	73	66	64	59
	S4	66.1	82	65	71	71	52	61	61
	S5	60.7	79	63	55	66	46	63	53
	S6	58.4	80	53	56	65	49	52	54
	S7	55.7	75	64	49	66	36	53	47
	S8	60.3	78	54	54	70	50	57	59
27B	S1	24.9	37	23	17	21	33	19	24
	S2	23.4	32	22	23	26	23	13	25
	S3	82.4	87	77	80	85	90	85	73
	S4	85.0	85	79	85	88	91	86	81
	S5	83.7	84	84	82	87	84	80	85
	S6	81.1	79	69	86	85	85	81	83
	S7	82.7	81	80	78	88	86	82	84
	S8	81.4	83	70	77	87	85	83	85

A.5.2 Example Prompt

This section reproduces the complete prompt template used for the simplest of the eight strategies—Strategy S1 (Minimal Prompt), described in Sec. 6.4.2 and summarized in Table 6.1. The full experiment using all eight strategies is discussed throughout Sec. 6.4.2–Sec. 6.4.4. The prompt is shown together with the expected single-digit model output for the sample task it illustrates. The remaining seven prompt templates (S2–S8) follow the same overall structure, differing only in the declarative content they inject (e.g., worked examples for S5–S6, chaining instructions for S7–S8); they are omitted here because including all of them in print would occupy excessive space, but are available in the accompanying code repository.

```
RAVEN'S PROGRESSIVE MATRICES (RPM):
A 3x3 grid puzzle with 8 context panels shown; the missing bottom-right panel
must be chosen from 8 answer candidates (numbered 0-7).

SPATIAL ARRANGEMENTS (7 types):
- center_single:                          1 object in center
- distribute_four:                        up to 4 objects at positions TL, TR, BL, BR
- distribute_nine:                        up to 9 objects at positions TL, TC, TR, ML, c, MR, BL, BC, BR
- in_center_single_out_center_single:     2 objects - outer single | inner single
- in_distribute_four_out_center_single:   outer single | up to 4 inner objects (TL, TR, BL, BR)
- left_center_single_right_center_single: 2 objects - left | right
- up_center_single_down_center_single:    2 objects - up | down

POSITION LABELS:
  c  = center
  TL = top-left,    TC = top-center,    TR = top-right
  ML = mid-left,    MR = mid-right
  BL = bottom-left, BC = bottom-center, BR = bottom-right

HOW PANELS ARE DESCRIBED:
- Single object (non-slotted layout):
    "large orange square"

- Two layouts separated by |  (outer | inner,  left | right,  up | down):
    "large white circle | [TL=big red triangle, BR=small blue circle]"
    "small green triangle | large azure pentagon"

- Slotted layout (distribute_four or distribute_nine):
    "[TL=large orange square, TR=small green circle, BL=medium pink pentagon]"
    A missing slot means that position is empty in this panel.

HOW ANSWER CANDIDATES ARE DESCRIBED:
- Single-layout arrangements (answer choices have one description):
    "[0] large orange square"
    "[0] [TL=large orange square, TR=small green circle]"
- Two-layout arrangements (both layouts always joined by | regardless of format):
    "[0] small green triangle | large azure pentagon"
    "[0] large white circle | [TL=big red triangle, BR=small blue circle]"

SEPARATOR HIERARCHY - three distinct levels:
  ','  separates objects WITHIN a panel
  ';'  separates panels WITHIN a row
  newline separates rows

IMPORTANT: Position is a rule-bearing attribute. A lone entity at [BR=...]
is different from one at [TL=...]. Track which slots are filled across each
row - the occupied position can itself follow a Constant, Progression, or
Distribute-Three rule.

ATTRIBUTE VALUES:
- Type:  triangle, square, pentagon, hexagon, circle
- Size:  tiny(0) < small(1) < medium(2) < big(3) < large(4) < huge(5)
- Color: white, azure, purple, pink, orange, olive, green, red, navy, black

RULES (apply ROW-WISE, independently to each attribute):
- CONSTANT:        same value in all three panels of a row
- PROGRESSION:     value increases or decreases by a fixed step (+1, +2, -1, -2) across panels
- ARITHMETIC:      third panel value = first ± second
- DISTRIBUTE_THREE: three distinct values, one per panel, distributed across the row

IMPORTANT NOTES:
- Rules apply row-wise, independently to each attribute
- Each attribute (Type, Size, Color, Number, Position) can follow a different rule
- Number and Position are coupled: if Number is non-Constant then Position must be
  Constant, and vice versa - they cannot both vary in the same row

PUZZLE:
Row 1: small black pentagon; big navy pentagon; huge red pentagon
Row 2: small olive triangle; big orange triangle; huge pink triangle
Row 3: tiny olive hexagon; medium orange hexagon
```

```
ANSWER CHOICES:
[0] large pink circle
[1] large azure circle
[2] large azure hexagon
[3] big pink hexagon
[4] big pink circle
[5] big azure hexagon
[6] large pink hexagon
[7] big azure circle

> CRITICAL INSTRUCTION:
Output a SINGLE DIGIT (0-7) and absolutely nothing else.
Do NOT write any explanation, reasoning, or commentary.
Any output other than a single digit 0-7 is an error.

Answer:
```

Expected model output:

```
6
```

Listing A.2: Example S1 (Minimal) prompt for an I-RAVEN task (see Sec. 3.2.3), together with the expected single-digit model output. This is the simplest of the eight prompt strategies and uses no few-shot examples or chain-of-thought scaffolding.

Poznan University of Technology
Faculty of Computing and Telecommunication
Institute of Computing Science

Typeset using LaTeX.