

Poznań University of Technology

Politechnika Poznańska



Faculty of Control, Robotics and Electrical Engineering
Institute of Robotics and Machine Intelligence

*Wydział Automatyki, Robotyki i Elektrotechniki
Instytut Robotyki i Inteligencji Maszynowej*

Doctoral dissertation

**Adaptive and distributed deep learning techniques
for reliable sensing on edge devices**

*Adaptacyjne i rozproszone techniki głębokiego uczenia
dla niezawodnej percepcji na urządzeniach brzegowych*

Mateusz Piechocki

Supervisor: Assoc. Prof. Tomasz Pajchrowski, PhD, DSc Eng.
Auxiliary supervisor: Marek Kraft, PhD Eng.

Poznań, September 2026

Mateusz Piechocki

Adaptive and distributed deep learning techniques for reliable sensing on edge devices

Doctoral dissertation, Poznań, September 2026

Supervisor: Assoc. Prof. Tomasz Pajchrowski, PhD, DSc Eng.

Auxiliary supervisor: Marek Kraft, PhD Eng.

Poznań University of Technology

Institute of Robotics and Machine Intelligence

Faculty of Control, Robotics and Electrical Engineering

Abstract

Deep learning underpins perception in robotics, industrial inspection, and smart infrastructure. These applications require low latency, operation under unstable connectivity, and local data processing, so inference is moving to edge devices placed close to the sensors. Such devices, however, have limited memory, compute, and energy, and the limited set of supported operations narrows the range of models that can execute at all. Even if these barriers are overcome, further challenges appear after deployment. Edge AI systems are expected to operate for years, and environmental variability means that they run under conditions that differ from those encountered during training. Data drift necessitates adaptation, which is, in turn, severely hindered by the aforementioned hardware constraints and weak supervisory signals.

This dissertation examines whether task-specific perception algorithms, hardware-aware execution, selective model updates, and compact knowledge exchange yield a better trade-off between predictive performance and the costs of inference, adaptation, and collaboration. The work follows the lifecycle of a perception system on an edge device, covering deployment, adaptation, and interaction across devices. For every task, quality metrics are set against performance figures measured on reference platforms ranging from microcontrollers to embedded GPUs. Three vision tasks serve this purpose, namely occupancy estimation in low-resolution thermal images, induction-motor fault diagnosis from thermal images, and short-horizon solar irradiance forecasting from all-sky imagery, the last of which is used to evaluate all three stages. The dissertation contributes a measured, lifecycle-wide validation that reliable edge perception can be sustained on devices with limited hardware resources. The evidence, gathered with a reporting framework of nine reliability criteria, confirms the hypothesis through its four components, verified in separate case studies. The dissertation also provides practical guidelines on when model-side optimization is beneficial, and when adaptation and collaborative learning justify their costs.

Streszczenie

Uczenie głębokie stanowi podstawę percepcji w robotyce, inspekcji przemysłowej i inteligentnej infrastrukturze. Zastosowania te wymagają niskich opóźnień, działania przy niestabilnej łączności oraz lokalnego przetwarzania danych, dlatego wnioskowanie przenosi się na urządzenia brzegowe umieszczone bezpośrednio przy czujnikach. Urządzenia te dysponują jednak ograniczoną pamięcią, mocą obliczeniową i energią, a wąski zestaw obsługiwanych operacji obliczeniowych zawęża zakres modeli, które można uruchomić. Nawet jeśli uda się pokonać te bariery, po etapie wdrożenia pojawiają się kolejne wyzwania. Od systemów brzegowej sztucznej inteligencji (ang. *edge AI*) oczekuje się niezawodnego działania przez lata, a zmienność środowiska sprawia, że operują one w warunkach odbiegających od tych z etapu uczenia. Zjawisko dryfu danych wymusza na modelach adaptację, która z kolei jest znacząco utrudniona ze względu na wspomniane bariery sprzętowe oraz słaby sygnał nadzorujący.

W rozprawie zbadano, czy algorytmy percepcji dostosowane do zadania, implementacja uwzględniająca specyfikę sprzętową, selektywne aktualizacje modelu oraz kompaktowa wymiana wiedzy pozwalają uzyskać korzystniejszy kompromis między jakością predykcji a kosztami wnioskowania, adaptacji i współpracy. Układ pracy odpowiada cyklowi operacji związanych z systemem percepcji na urządzeniu brzegowym, obejmując wdrożenie, adaptację i współdziałanie między urządzeniami. Dla każdego zadania wskaźniki jakościowe zestawiono z parametrami wydajnościowymi zmierzonymi na platformach referencyjnych, od mikrokontrolerów po procesory graficzne dla systemów wbudowanych. Do tego celu wykorzystano trzy zadania wizyjne: szacowanie liczby osób na podstawie obrazów termowizyjnych niskiej rozdzielczości, diagnostykę uszkodzeń silników indukcyjnych z wykorzystaniem obrazów termowizyjnych oraz prognozowanie natężenia promieniowania słonecznego w krótkim horyzoncie czasowym w oparciu o obrazy całego nieba. Ostatnie z nich posłużyło do oceny wszystkich trzech etapów. Wkładem rozprawy jest weryfikacja możliwości realizacji niezawodnej percepcji na urządzeniach o ograniczonych zasobach, obejmująca pełny cykl operacyjny i oparta na rzeczywistych pomiarach kosztów. Uzyskane wyniki, zestawione według schematu raportowania opartego na dziewięciu kryteriach niezawodności, potwierdzają postawioną hipotezę, której cztery składowe zweryfikowano w odrębnych studiach przypadku. Rozprawa dostarcza również praktycznych wytycznych wskazujących, kiedy optymalizacja po stronie modelu przynosi korzyści oraz kiedy koszty adaptacji i uczenia zespołowego są uzasadnione.

Acknowledgments

I am sincerely grateful to my supervisor, Professor Tomasz Pajchrowski, for his guidance, support, and thoughtful feedback throughout this work. I also thank my auxiliary supervisor, Dr. Marek Kraft, for his friendliness, practical advice, and encouragement at key stages, as well as for always being willing to help. I am also grateful to Dr. Alessandro Capotondi from the University of Modena and Reggio Emilia, whose collaboration and valuable insights enriched several parts of this dissertation.

I am grateful to my lab colleagues for the many conversations, their help whenever I needed it, and the collaborations that influenced several parts of this thesis. I also thank the administrative and technical staff at Poznań University of Technology for their support throughout my doctoral studies.

I owe my parents heartfelt gratitude for giving me an early foundation of curiosity, discipline, and perseverance, and for their patience, support, and unwavering confidence in me over the years that led me to this point. And to my siblings, for being exactly that in every sense, and for not asking too many times when the dissertation would finally be finished.

Above all, I thank my wonderful wife, Kornelia. You carried me through every deadline and every long evening, and you believed in me even when I did not. Thank you for your patience, your love, and for making home a place I could always come back to. I could not have done this without you.

Contents

Abstract	III
Streszczenie	V
Acknowledgments	VII
List of acronyms	XIII
1 Introduction	1
1.1 Problem description and motivation	1
1.2 Research hypothesis, questions, and objectives	2
1.3 Impact of the research	4
1.3.1 Scientific contribution	4
1.3.2 Societal relevance and potential impact	5
1.4 Thesis structure	6
2 Background	7
2.1 Edge computing	7
2.1.1 Device classes	7
2.1.2 The constraints that govern deployability	8
2.2 Deep learning for sensor data	9
2.2.1 Architectures	9
2.2.2 From sensor streams to model inputs	10
2.2.3 From trained model to deployable model	11
2.3 Reliability of deployed inputs and evidence	12
2.3.1 Physical consistency and shortcut learning	12
2.3.2 Input-level failures and integrity violations	12
2.3.3 Input admissibility for inference and adaptation	13
2.4 On-device adaptation	13
2.4.1 Why static models degrade after deployment	13
2.4.2 A formal model of on-device adaptation	14
2.4.3 Supervision regimes	15
2.4.4 What is updated, and parameter-efficient adaptation	15
2.4.5 When to update, triggers and gating	16
2.4.6 Failure modes of on-device adaptation	16

2.5	Collaboration across devices	17
2.5.1	Federated averaging and its failure under structured heterogeneity	17
2.5.2	Design responses and the exchanged object	17
2.6	Positioning and the research gap	18
2.6.1	Reliability criteria by lifecycle stage	18
2.6.2	Position taken in this thesis	19
2.6.3	Research gap and mapping to the research questions	20
2.7	Experimental framework	22
2.7.1	Assessment logic	22
2.7.2	Sensing workloads and datasets	22
2.7.3	Reference edge platforms	24
2.7.4	Task families and task-level metrics	24
2.7.5	System-level metrics and measurement procedure	28
2.7.6	Reporting conventions	29
3	Reliable sensing and inference on the edge	31
3.1	Deployment criteria and evaluation protocol	31
3.2	Deployment feasibility under kilobyte memory budgets	32
3.2.1	Method	33
3.2.2	Evaluation	36
3.2.3	Conclusions	39
3.3	Deployment feasibility and physical consistency in induction motor fault diagnosis	40
3.3.1	Method	40
3.3.2	Evaluation	42
3.3.3	Conclusions	46
3.4	Predictive robustness and its limits in irradiance forecasting	47
3.4.1	Method	47
3.4.2	Evaluation	50
3.4.3	Conclusions	58
3.5	Summary and the limits of static deployment	59
4	Reliable adaptation on the deployed device	61
4.1	Adaptation protocol and evaluation setup	61
4.1.1	The adaptation loop	61
4.1.2	Encoder-head factorization and design for stability	62
4.2	Error-triggered updates under lagged supervision	64
4.2.1	Method	65
4.2.2	Evaluation	67
4.2.3	Conclusions	76
4.3	Input admissibility, detecting faults and testing exclusion	76

4.3.1	Method	77
4.3.2	Evaluation	79
4.3.3	Conclusions	83
4.4	Update affordability through accelerator-backed execution	83
4.4.1	Method	84
4.4.2	Evaluation	86
4.4.3	Conclusions	89
4.5	Summary and the limits of local adaptation	90
5	Communication-limited collaboration across deployed devices	93
5.1	Collaboration protocol and evaluation setup	93
5.1.1	What a single device cannot learn	93
5.1.2	Evaluation setup	94
5.2	Regime-stratified prototype exchange for multi-site forecasting	95
5.2.1	Method	95
5.2.2	Evaluation	99
5.2.3	Conclusions	107
5.3	Summary	107
6	Conclusions	109
6.1	Answers to the research questions	109
6.2	Contributions	111
6.3	Limitations	112
6.4	Future work	113
6.5	Ethics statement	114
6.6	Statement on the use of generative AI	114
6.7	Declaration of research artifacts availability	114
A	List of publications	117
B	List of grants and awards	121
C	List of research projects	123
	List of figures	125
	List of tables	127
	Bibliography	129

List of acronyms

BESS	battery energy storage system
BMS	building management system
CNN	convolutional neural network
CPU	central processing unit
CUDA	Compute Unified Device Architecture
DRAM	dynamic random-access memory
Edge AI	edge artificial intelligence
FL	federated learning
FLOP	floating-point operation
FP16	half-precision floating-point
FP32	single-precision floating-point
FS	forecast skill
GDPR	General Data Protection Regulation
GPU	graphics processing unit
HVAC	heating, ventilation, air conditioning
IEP	image enhancement pipeline
IID	independent and identically distributed
INT8	8-bit integer
MAC	multiply–accumulate operation
MAE	mean absolute error
MAPE	mean absolute percentage error
MCU	microcontroller unit

ML machine learning

MSE mean squared error

NCS2 Neural Compute Stick 2

NPU neural processing unit

NRBR normalized red-blue ratio

nRMSE normalized root mean squared error

ONNX Open Neural Network Exchange

OpenVINO Open Visual Inference and Neural Network Optimization

ORT ONNX Runtime

RAM random-access memory

RBR red-to-blue ratio

RGB red, green, blue

RMSE root mean squared error

RPI Raspberry Pi

RSS resident set size

SBC single-board computer

SRAM static random-access memory

TFLite TensorFlow Lite

TFLM TensorFlow Lite Micro

TPU tensor processing unit

USB Universal Serial Bus

VPU vision processing unit

VRAM video random-access memory

Introduction

Deep learning forms the foundation of contemporary perception pipelines in robotics [1], industrial inspection [2], and smart infrastructure [3]. In these domains, perception systems must be fast enough to support closed-loop control, resilient under changing conditions, and reliable despite limited or intermittent connectivity. Taken together, these demands are driving a shift away from centralized cloud architectures toward resource-limited edge devices placed directly alongside the sensors [4].

This dissertation studies efficient and reliable sensing on edge devices through task-specific pipelines, hardware-aware execution, selective model updates, and compact knowledge exchange. The experimental work focuses predominantly on visual sensing and image processing, including low-resolution thermal occupancy estimation, thermal-image motor-state classification, and all-sky-image solar irradiance forecasting supplemented by irradiance measurements. The component studies relate predictive performance to the measured costs of inference, adaptation, and communication. The nine criteria in Sec. 2.6.1 organize the evidence on deployment, adaptation, and collaboration.

1.1 Problem description and motivation

Centralized operations and lifecycle management of deep learning systems are effective when network connectivity is stable, data transfer is limited, and computing time is not a major constraint. Continuous sensing applications that demand low latency, privacy, and reliable operation under intermittent connectivity often cannot be served this way, which motivates edge processing that brings perception closer to the data source. However, the deployment of deep learning models on edge devices is challenging. Constrained memory, limited computing power, and energy consumption limit real-time performance, and varied hardware and software stacks require configuration-specific optimization [4] of the entire signal-processing pipeline rather than of the model alone.

Meeting operational requirements at deployment does not guarantee long-term reliability. Edge systems typically have extended lifespans and operate under nonstationary conditions, so configurations that are sufficient at installation may become inadequate over time [5]. Static models can deteriorate under distribution shift, while adaptation introduces computational overhead and can itself degrade predictions. Updates may rely on delayed or missing supervision, and an unusual

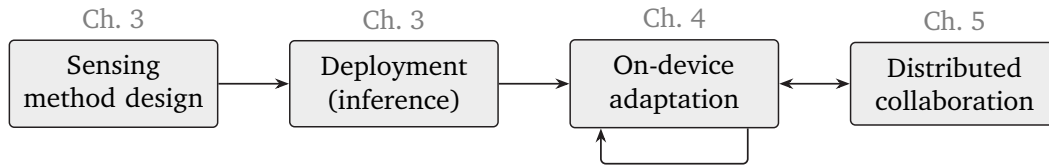


Fig. 1.1. The edge-device perception lifecycle considered in this dissertation. The studies address offline method design, deployment and on-device inference, local adaptation, and collaborative exchange. The arrows show how the individual stages affect each other.

image is not necessarily unsuitable training data [6]. Sustained reliability therefore requires effective update methods and explicit policies on whether and when to adapt. Moreover, frequently sensing systems are distributed across multiple devices, with similar applications deployed in different environments and configurations. Knowledge gained from one deployment can inform others operating under comparable conditions. Collaborative learning facilitates knowledge transfer without the need to share raw data [7]. For resource-limited devices, determining which information to transmit is still a major challenge for current aggregation methods. The edge-perception lifecycle diagram in Figure 1.1 illustrates the connections among offline design, deployment, local adaptation, and knowledge exchange. Although these stages inform the research questions in Sec. 1.2, not all mechanisms were evaluated within a single integrated system.

1.2 Research hypothesis, questions, and objectives

This dissertation approaches efficient sensing on an edge device as a co-design challenge that integrates sensing modalities and input representations with execution architecture, runtime behavior, adaptation schedules, and mechanisms for communication and knowledge sharing, which together form the basis for the central hypothesis.

Task-specific sensing pipelines, hardware-aware execution, selective model updates, and compact knowledge exchange can support sensing on edge devices by improving the trade-off between predictive performance and the costs of inference, adaptation, and collaboration.

The hypothesis is evaluated through component comparisons against the reference methods in each study. Evidence for efficiency, such as reduced execution cost or communication payload, is interpreted alongside predictive performance and within the relevant measurement boundary. Reliability is analyzed using the nine criteria defined in Sec. 2.6.1, while the comparison rules and available evidence are specified in Sec. 2.7.

The four research questions and their objectives cover distinct aspects of pipeline design, execution, update policy, and knowledge exchange within the sensing studies.

1. **RQ1.** How do task-specific sensing pipelines influence predictive performance and system complexity, and what evidence supports their physical consistency and robustness?
O1. Develop and evaluate tailored data workflows. Compare their predictive performance and the complexity of models with relevant scientific studies, inspecting the models' physical reliability where possible.
2. **RQ2.** How do architecture, numerical precision, and hardware-aware execution impact inference and on-device adaptation?
O2. Implement and characterize deep learning model optimization techniques, including any conversion failures and metric losses. Measure their impact on predictive performance and system overheads.
3. **RQ3.** How do selective model updates with delayed supervision affect predictive performance and update cost compared with static inference and other adaptation policies?
O3. Develop and evaluate a selective update policy under delayed supervision, comparing predictive outcomes with static and reference adaptation approaches. Measure the current policy's update cost and propose techniques to accelerate this process. Test whether image-fault detection justifies excluding observations from adaptation, and identify the remaining evidence needed for input admissibility, update justification, and affordability.
4. **RQ4.** How does knowledge sharing affect predictive performance when compared with isolated local adaptation across heterogeneous sites?
O4. Develop and evaluate knowledge exchange methods. Compare predictive outcomes with isolated local adaptation, quantify communication payload and execution costs, and examine predictive preservation, communication efficiency, and participation resilience while keeping raw observations local.

Tab. 1.1. Research objectives, workloads, and evidence aligned and cross-mapped. An extensive review of this alignment is provided in Table 6.1.

Obj.	Focus	Workload	Evidence
O1	Task-specific pipelines and their predictive performance and complexity trade-offs <i>Stage: design; deployment</i>	Thermal occupancy estimation; induction motor fault diagnosis; solar irradiance forecasting with physically structured inputs (Ch. 3)	Predictive and model-size comparisons; qualitative physical-consistency evidence; limited cross-site robustness.
O2	Hardware-aware inference and on-device training <i>Stage: deployment; update</i>	Thermal occupancy estimation; induction motor fault diagnosis (Ch. 3); hardware-accelerated adaptation pipeline (Ch. 4)	Execution feasibility, measured costs, conversion failures and quantization effects on the evaluated stacks; separate accelerator-backed on-device adaptation study.
O3	Selective updates under delayed supervision <i>Stage: update</i>	Solar irradiance forecasting under domain shift; hardware-accelerated adaptation pipeline (Ch. 4)	Policy evaluation with quantified per-action update cost on two boards; hardware-accelerated adaptation assessment.
O4	Compact knowledge exchange compared with static model and local adaptation <i>Stage: collaboration</i>	Multi-site solar irradiance forecasting (Ch. 5)	Payload size quantification across exchange schemes; own-prototype control; payload and controlled dropout comparisons; matched per-client hardware overhead.

1.3 Impact of the research

This dissertation advances efficient, reliable edge-device sensing by designing resource-aware perception methods, implementing them on embedded hardware, and validating them using data-driven measurements. It is organized according to deployment, adaptation, and exchange, and evaluates each design choice for its impact on predictive and operational factors. Although the main emphasis is on the scientific contribution, societal relevance is discussed through technical requirements and modeled indicators, since extensive field testing is beyond the thesis scope.

1.3.1 Scientific contribution

Methodological advancements. The primary contribution of this thesis is a measured, lifecycle-wide demonstration that reliable perception can be sustained on constrained devices, produced with a reporting framework of nine reliability criteria and organized along the three linked stages of deployment, adaptation, and collaboration, for each of which a methodological contribution is proposed. Case studies involving compact model architectures for occupancy estimation, induction motor fault classification, and short-term solar irradiance forecasting illustrate how the incorporation of domain knowledge can decrease the required model capacity

for a given sensing task. An error-triggered update policy is proposed to support on-device adaptation of forecasting models when ground-truth labels can be obtained after a temporal lag. An empirical study of a training pipeline supported by a commodity edge artificial intelligence (Edge AI) accelerator outlines the potential for accelerating on-device learning. A collaborative learning approach based on a regime-stratified prototype exchange scheme is introduced as a communication-efficient alternative to transmitting full model parameters or states in a multi-site setting.

Research artifacts and datasets. The research supplies a thermal-image dataset for induction-motor fault diagnosis and public source code, configurations, and trained weights for the studies listed in Sec. 6.7. Two all-sky monitoring stations in Poland recorded paired sky images and irradiance measurements over a full year of operation. A subset is already public as part of Sec. 3.4; release of the complete station records is planned.

1.3.2 Societal relevance and potential impact

Privacy and data minimization. Every configuration studied keeps inference on the device, so raw observations are processed locally rather than sent to a remote service. The occupancy pipeline uses low-resolution thermal sensing, which limits directly identifying visual detail, and the collaboration study exchanges learned summaries in place of raw sensor data, with roughly a thousand times fewer values than full-model exchange. These choices comply with the data-minimization principle without formalizing privacy protection. Residual exposure is addressed in Sec. 6.5.

Accessibility of deployment. The assessed setups show that the chosen sensing pipelines can execute on microcontroller units (MCUs) and single-board computers (SBCs). The occupancy study reports board-level daily energy estimates for a defined sensing and inference schedule. Local supervised adaptation can be performed when labels arrive with delay, and a compact exchange format lowers bandwidth relative to full-state communication. Together, these outcomes indicate that local, on-device operation is feasible, also under limited connectivity. Nonetheless, they do not demonstrate fully unsupervised unattended operation.

Sustainability and renewable integration. Short-horizon irradiance forecasts support ramp management and storage dispatch at photovoltaic plants [8]. Low-resolution thermal occupancy estimates can inform ventilation demand, equipment schedules, and other occupancy-aware heating, ventilation, air conditioning (HVAC) decisions in smart buildings [9]. Thermal-image motor-state classification can support condition monitoring, inspection prioritization, and predictive-maintenance workflows [10].

1.4 Thesis structure

The dissertation is divided into six chapters aligned with the perception lifecycle diagram in Fig. 1.1 and the objectives outlined in Sec. 1.2, with the solar irradiance forecasting application encompassing all stages.

Chapter 2 introduces the basic concepts for reliability in Edge AI and discusses edge device design, deployability constraints, deep learning for sensor data, input integrity, and the shift toward on-device adaptation. This background identifies the research gap the dissertation addresses. The chapter then closes with the experimental framework shared across the empirical chapters.

Chapter 3 formalizes deployment reliability through three requirements, namely deployment feasibility, physical consistency, and predictive robustness, illustrated by case studies on thermal occupancy estimation, induction-motor diagnosis, and intra-hour solar irradiance forecasting. The third case study leads to expanded exploration in subsequent chapters.

Chapter 4 first evaluates error-triggered adaptation under delayed supervision, then tests whether image-fault screening helps that policy, and finally reports a separate hardware-accelerated on-device adaptation study. The chapter assesses the adaptation criteria of input admissibility, update justification, and affordability together with the evidence available for each component.

Chapter 5 evaluates regime-stratified prototype exchange for irradiance forecasting across heterogeneous sites. Comparisons with isolated adaptation, own-prototype regularization, and parameter exchange separate the value of compact communication from the site-dependent peer increment.

Chapter 6 summarizes the key findings, contributions, limitations, and future directions, and adds an ethics statement and a record of the datasets, artifacts, and source code related to the doctoral research.

Additionally, Appendices A, B, and C document the scientific activities from the doctoral program (2021–2026), including publications, research grants, and project involvement.

Overall, this dissertation draws on the author’s publications cited below. Sections 3.2 to 3.4 build on [11, 12, 13, 14, 15], and Secs. 4.2 to 4.4 build on [16, 17, 18]. In these publications, the author reports leading the experimental design, implementation, measurement, and writing.

Background

This chapter provides the foundation for the experimental chapters. It reviews the background and related literature covering the perception lifecycle in Fig. 1.1, including edge computing, deep learning for sensor data, the reliability of deployed inputs and evidence, post-deployment model adaptation, and collaboration across a fleet (Secs. 2.1 to 2.5). Section 2.6 states the reliability criteria, the position taken in this thesis, and the research gap. The chapter concludes with Sec. 2.7, which describes the reporting framework common to the empirical chapters.

2.1 Edge computing

The edge computing paradigm brings processing and storage close to the data source, where decisions and actions are taken [4, 19, 20, 21]. Chapter 1 motivated this shift through latency, connectivity, and data governance. Thus, this section narrows the scope to deployability. Cloud analytics benefits from central control. Operators can expand compute resources, apply software patches, retrain models, and deploy updates from one place. Edge deployments operate under different conditions. The processing pipeline has to run within strict compute, memory, and energy limits determined by hardware already deployed in the field [5]. Connectivity can be unreliable, reducing the ability to depend on remote services and increasing the requirement for dependable on-device failure handling. These constraints affect more than raw throughput. They shape data pre-processing, model design, and numerical precision, as well as latency requirements. A deployable system should also preserve consistent prediction performance as conditions vary and keep resource consumption predictable over long operating periods. Given these requirements, the characteristics of the edge device become a key design input for the overall system.

2.1.1 Device classes

Edge devices range from sensor nodes with kilobytes of memory to compact field computers. For this dissertation, four classes of Edge AI hardware can be distinguished, starting with MCU-based devices addressing the TinyML regime, SBCs, heterogeneous SBC–accelerator systems, and ending with embedded graphics processing unit (GPU) platforms. Their differences go beyond throughput. On MCU-class devices, memory allocation is fixed at compile time and execution is bare-metal [22, 23]. SBCs are limited to central processing unit (CPU) throughput when processing deep learning pipelines. Heterogeneous configurations resolve

Tab. 2.1. Edge device categories and their dominant constraints. Memory values provide an order-of-magnitude characterization of a class, not a specification for any particular board. Selected platforms and their execution styles are given in Table 2.6.

Device category	Typical memory scale	Dominant constraints
MCU-class	~64 kB–1 MB SRAM	1. Memory (SRAM) capacity 2. Limited instruction and operator support
SBC (CPU-only)	~1 GB–8 GB DRAM	1. CPU throughput 2. Memory (DRAM) bandwidth
Heterogeneous (SBC + accelerator)	Host DRAM plus constrained accelerator memory	1. Host–accelerator data-transfer overhead 2. Limited operator and data precision support
Embedded GPU	~2 GB–64 GB unified memory	1. Memory (VRAM) bandwidth 2. Power-mode throughput ceiling

this issue by delegating part of the computations to high-performance accelerators. However, such systems introduce a host–accelerator interface, making operator support and data movement design constraints. In turn, embedded GPU platforms deliver high computing power, but at the cost of higher energy consumption and device cost. Memory, compute, and power span orders of magnitude across these classes, determining which perception methods are feasible independently of predictive performance. Table 2.1 contains a relevant comparison of these categories, characterizing them and listing their dominant constraints.

2.1.2 The constraints that govern deployability

Deployability is rarely determined by model predictive performance alone. Rather, it depends on how an end-to-end sensing-to-inference pipeline interacts with the target device’s resources and runtime. The following constraints determine whether a pipeline can be deployed on a given device category, and they are usually met in the order listed.

1. **Memory.** This is often the predominant constraint. In many cases, activation memory requirements, rather than parameter storage, constitute the primary bottleneck. As a result, a network that is compact enough to fit in memory may still be too large to run [24].
2. **Latency.** A property of the full decision loop rather than of the forward pass alone, since input formation, memory operations, host–accelerator transfers, and postprocessing share the sampling-interval budget. For the sequential pipelines in this thesis, the end-to-end inference latency is decomposed as in Eq. 2.1.

$$T_{\text{e2e}} = T_{\text{pre}} + T_{\text{infer}} + T_{\text{post}}, \quad (2.1)$$

where the three terms partition the time from the moment a frame is available in the acquisition buffer to the moment a decision is available. Sensor readout and the acquisition buffer itself lie before T_{pre} and are not part of the sum. T_{pre} covers input formation, from reading the latest frame or window out of the acquisition buffer, through the memory copies, data transforms, and construction of any derived channels, to the loaded input tensor in host memory. T_{infer} is the runtime invocation from call to return, and on heterogeneous platforms it includes the transfer of the input tensor to the accelerator and of the output tensor back to the host. T_{post} maps raw outputs to a decision.

3. **Toolchain and operator support.** Deployability is constrained by whether the model graph can be compiled for the target runtime. Some stacks allow partial mapping by running unsupported operators on the host CPU, but this option is not always available, and it is often suboptimal for response time and energy consumption.
4. **Numerical and interface heterogeneity.** Acceptable precisions (single-precision floating-point (FP32), half-precision floating-point (FP16), 8-bit integer (INT8)) and the programming interfaces differ across CPUs, embedded GPUs, and specialized hardware.
5. **Energy and thermal limits.** While these constraints rarely stop a single inference, they can restrict continuous operations and ultimately limit long-term throughput [25].

2.2 Deep learning for sensor data

Deep learning extracts semantic information from sensor streams by learning hierarchical representations [26]. On a device, those representations must also fit a continuous processing pipeline and the constraints specified in Sec. 2.1.2. The following subsections connect architectures and model inputs, including inputs that carry physical knowledge, to predictive robustness before considering deployment. They also highlight the practical trade-offs that arise when these concerns interact with resource budgets.

2.2.1 Architectures

For images and low-resolution sensor arrays, such as thermal or time-of-flight imagers, convolutional neural networks (CNNs) build spatial hierarchies that progress from local edges and textures to object-level structure. ResNet backbones [27] remain a standard reference, but they are often too costly for MCU-class devices. MobileNets reduce parameters and multiply-accumulate operations with depthwise separable convolutions and inverted residual blocks [28, 29], whereas EfficientNet scales depth, width, and input resolution jointly through a single compound coefficient [30]. Hybrid designs keep convolutions within high-resolution

stages and apply attention only to coarse feature maps [31]. However, the number of parameters and floating-point operations (FLOPs) are incomplete proxies for deployability. Peak activation memory frequently dominates on devices with a few hundred kilobytes of SRAM [32], and a model is usable only if the target runtime supports its operators [22]. Moreover, attention blocks and layers with large-magnitude outlier activations tend to lose more predictive performance under 8-bit quantization than standard convolutional layers [33]. When labeled data are scarce, an encoder pretrained on a large general-purpose dataset such as ImageNet can be adapted to the target task [34], a setting evaluated in Sec. 3.4.

This separation into a reusable feature extractor and a task-specific head is used explicitly throughout the thesis. Most models considered here can be written as the composition

$$f(x; \theta_b, \theta_h) = h(g(x; \theta_b); \theta_h), \quad (2.2)$$

where the encoder g maps an input x to an embedding $z = g(x; \theta_b) \in \mathbb{R}^d$, and the head h maps this embedding to the task output. In typical edge-device-oriented models, g holds most of the parameters and computation, while h is comparatively small, often a single linear or shallow layer. The embedding z therefore serves as a common interface for the input validation and collaboration mechanisms introduced later. Its width d determines how compact the stored or transmitted representation is. Still, it does not reduce the cost of producing z , since running and training the encoder carry their own computation and memory costs [35].

2.2.2 From sensor streams to model inputs

A model observes a physical process only through a limited temporal view, such as one frame per decision for vision or a window of recent samples for continuous signals. Combining images with time-aligned scalar measurements additionally requires fusion. Early fusion concatenates the modalities before an encoder, whereas intermediate fusion encodes only the high-dimensional modality and appends the scalar measurements to its embedding [36, 37]. Unlike late fusion, which runs one encoder per modality, these schemes use a single encoder. However, it is not the only cost between sensor and decision. Forming the inputs described above is part of preprocessing, alongside normalization, compression, and domain-specific feature extraction, while postprocessing, such as thresholding and morphological filtering, turns model outputs into decisions. Both stages add their own processing time and energy, and enter Eq. 2.1 as T_{pre} and T_{post} .

When the deployment domain offers a physical or geometric model of the sensed process, part of what the network would otherwise have to infer from data can be computed in advance [38, 39]; such values may be fixed by time and position, as the position of the Sun is for a given site, by a physics-based transformation of the raw

measurements, or by contextual sensors that provide information the imagery does not contain. They enter the single encoder through the fusion schemes mentioned above, as extra input channels or as values concatenated to the embedding, and how strongly the network relies on them is learned. Since they are based on a physical model rather than the training distribution, they can be applied to a different site or season as long as such a model remains valid.

2.2.3 From trained model to deployable model

A trained model is not yet a deployable one. State-of-the-art architectures are often not designed for edge deployment and exceed the budgets of Sec. 2.1.2. Therefore, a model optimization and compilation step separates the trained network from the deployed artifact.

Export and compilation. Models are exported to an intermediate representation, Open Neural Network Exchange (ONNX) in this thesis [40], before graph optimization and target-specific compilation. Targets outside the ONNX ecosystem require conversion to their own runtime format, as with tensor processing unit (TPU) delegation through TensorFlow Lite (TFLite) [41] and STM32 execution through CubeMX with X-CUBE-AI. Compilation and execution take three forms.

1. **Static bare-metal execution (MCUs).** The model is compiled by a platform-specific toolchain, from ONNX or via TensorFlow Lite Micro (TFLM) [42]. Because SRAM is highly limited, the runtime relies on an arena allocator, with peak activation and weight memory determined at compile time.
2. **Static CPU/GPU execution (SBCs and embedded GPUs).** The exported ONNX graph runs as one monolithic unit in ONNX Runtime (ORT) through the CPU or Compute Unified Device Architecture (CUDA) execution provider, unless stated otherwise.
3. **Subgraph delegation (host + AI accelerator).** For neural processing units (NPU)s like the Hailo-8L, Google Coral, or Intel Neural Compute Stick 2 (NCS2), a whole model graph is deployed onto accelerator. When it is not possible due to hardware or its limited toolchain support, the runtime splits the graph so that subgraphs made up of supported operators are compiled into accelerator binaries, while the rest runs on the host CPU.

Quantization and restoration. In compact architectures, the main reduction is numerical precision, from FP32 to FP16 or INT8, which reduces the memory of parameters and activation buffers and, on supported hardware, the time and energy of integer arithmetic. The mapping to integer ranges is typically calibrated post-training on a small representative sample, which is inexpensive but sensitive to outliers. Architectures are affected unequally. Activation functions with wide output ranges and attention mechanisms struggle with single integer scales and lose

more than plain convolutional networks. Restoration techniques recover part of the loss, from inexpensive statistical corrections that rescale channels or correct bias shifts [43, 44] to data-driven procedures that fine-tune the quantized network against its full-precision counterpart [45, 46]. Compressing the encoder of Eq. 2.2 also alters the embedding, so a quantized encoder must be treated as a different embedding source from its full-precision counterpart. The export, compilation, and quantization configurations examined in this thesis are stated in Sec. 2.6.2.

2.3 Reliability of deployed inputs and evidence

A task metric alone does not establish whether a deployed input is suitable for inference or learning, nor whether the evidence a model uses corresponds to the sensed process. Four cases arise. Model evidence can rest on acquisition artifacts instead of on the physical mechanism (Sec. 2.3.1); sensing-path degradation may make an observation misleading or unusable (Sec. 2.3.2); a valid but unfamiliar input can expose model shortcomings (Sec. 2.3.3); and a persistent distribution shift can motivate adaptation (Sec. 2.4). These cases need different responses, but their distinction must itself be evaluated.

2.3.1 Physical consistency and shortcut learning

Providing physical structure in the inputs (Sec. 2.2.2) does not guarantee that a network will actually rely on it, and strong held-out performance does not indicate what drove a given prediction. In controlled data acquisition, background factors can correlate with the label as strongly as the underlying sensed phenomenon, and a network may rely on whichever cue is easiest to extract [47, 48]. Predictive metrics alone therefore cannot establish whether a model is physically consistent, that is, whether the evidence it uses matches an independent description of the sensed process. Attribution techniques approximate this evidence via perturbation-based analyses or gradients and have revealed such correlates in practice. Still, they lack a universal evaluation criterion and may be sensitive to the chosen perturbations and hyperparameters [49, 50, 51]. As a result, an acceptance gate prior to deployment would require an independently derived physical expectation and a reproducible decision rule, rather than relying on a visual plausibility check.

2.3.2 Input-level failures and integrity violations

Integrity failures originate in the sensing path rather than in the process being observed. Progressive degradation includes contamination, condensation, dead pixels, thermal noise, and sensor drift; transient failures include occlusion, motion blur, dropped frames, timestamp errors, and partial writes. Such failures can weaken or break the input–target relationship, but their presence does not imply that every

affected sample contains no useful information. A noisy but interpretable observation may call for a noise-tolerant predictor; an unusable or inconsistent observation may call for rejection. Learning from the latter can propagate a sensing error into model parameters. A screening rule therefore needs an explicit operating point and measurements of both missed corruptions and rejected valid observations.

2.3.3 Input admissibility for inference and adaptation

In this thesis, admissible denotes an input judged suitable for inference or learning, not decision-theoretic non-dominance. Model-agnostic checks include exposure and blur statistics, dropped-frame counters, and timestamp validation. Embedding-based distances, densities, and isolation scores [52, 53] complement these checks by detecting unfamiliar representations. When the deployed forward pass exposes the required embedding, scoring it avoids another encoder pass; its cost still depends on embedding width, reference-set size, and the scoring method. Repeated stochastic inference and ensembles require additional computation, whose feasibility depends on the device budget rather than being excluded for every streaming deployment.

Whatever the score, an exclusion rule requires a calibrated operating point whose consequences depend on the task. A false alarm can withhold a nominal observation and reduce coverage, while a missed fault leaves an affected observation in the stream. Neither outcome alone determines learning harm. An image fault may impair inference yet remain useful for supervised adaptation. The operating point must therefore be assessed through downstream prediction and update outcomes, including the information and execution costs of exclusion, alongside detection and flag rates. The screening configuration examined in this thesis is stated in Sec. 2.6.2.

2.4 On-device adaptation

Adaptation can recover useful performance when operating conditions degrade a fixed model. On a device, updates share the inference budget, supervision may arrive late, and learning from unsuitable data can leave persistent damage (Sec. 2.3.2). The following formalism connects these topics through three choices. These are what drive an update, which parameters change, and when the update occurs.

2.4.1 Why static models degrade after deployment

Static machine learning (ML) models face dynamic, non-stationary environments [7, 5]. Visual shift can arise from illumination, weather, seasonality, viewpoint, lens aging, or changes to compression and exposure control. These mechanisms regularly co-occur. Aging affects both input appearance and calibration, while weather

adds fast excursions to slow seasonal change. Relocation can change several distributions at once. Where observations remain valid measurements, these shifts call for adaptation instead of rejection, appropriate to integrity failures (Sec. 2.3.2).

2.4.2 A formal model of on-device adaptation

The parameters modified, the evidence used, and the update timing determine both adaptation behavior and resource cost. A common formalism makes these choices explicit. Considering a deployed predictor $f(\cdot; \theta)$ with parameters $\theta \in \mathbb{R}^P$, producing at discrete time t an estimate $\hat{y}_t = f(x_t; \theta_t)$ of a quantity y_t . In a static deployment $\theta_t \equiv \theta_0$ for all t . Adaptation replaces this with a parameter update law that maps the current parameters and the evidence available at time t to a new parameter vector:

$$\theta_{t+1} = \mathcal{A}(\theta_t, \mathcal{B}_t, \gamma_t), \quad (2.3)$$

where $\mathcal{B}_t$ denotes the evidence buffer (the samples and, where available, the measured targets, admissible for learning at time t) and $\gamma_t \in \{0, 1\}$ is a trigger determining whether an update is performed at all. Setting $\gamma_t \equiv 0$ recovers static deployment.

In the settings considered here, the operator $\mathcal{A}$ of Eq. 2.3 is realized by first-order optimization restricted to a designated subset of the parameters. Writing $\Pi_{\mathcal{S}}$ for the orthogonal projection onto a trainable subspace $\mathcal{S} \subseteq \mathbb{R}^P$, the update law takes the concrete form

$$\theta_{t+1} = \theta_t - \gamma_t \eta_t \Pi_{\mathcal{S}_t} \nabla_{\theta} \mathcal{L}(\theta_t; \mathcal{B}_t), \quad (2.4)$$

with learning rate η_t and objective $\mathcal{L}$. Equation 2.4 makes explicit the three design choices that distinguish adaptation mechanisms from one another, and these are treated as independent axes throughout:

1. **The trainable subspace $\mathcal{S}_t$** determines what changes. Its dimension governs the memory and compute cost of the update, because backpropagation needs only traverse the portion of the network on which $\Pi_{\mathcal{S}_t}$ acts non-trivially, and the activations of the remainder need not be retained.
2. **The objective $\mathcal{L}$ and buffer $\mathcal{B}_t$** determine what drives the change. Supervised adaptation requires measured targets and is consequently constrained by their availability and timeliness, whereas unsupervised adaptation substitutes a surrogate computable from inputs alone.
3. **The trigger γ_t** determines when change occurs. It is a control decision that governs update frequency and can enforce a specified resource or screening rule. Whether that rule avoids harmful updates must be established by evaluation.

Tab. 2.2. Adaptation method families expressed as instances of the update law of Eq. 2.4. Here ψ denotes frozen encoder parameters and ϕ trainable head or adapter parameters, so that $\theta = (\psi, \phi)$.

Family	Subspace $\mathcal{S}_t$	Objective $\mathcal{L}$, buffer $\mathcal{B}_t$	Trigger γ_t	Update cost
Full fine-tuning	$\mathbb{R}^P$	supervised, labeled buffer	scheduled	high
PEFT / head-only	$\{0\} \times \mathbb{R}^{ \phi }$	supervised, labeled buffer	scheduled	low
Test-time adaptation	often BN affine params	surrogate such as entropy, current sample only	every sample	medium
Online learning under delayed supervision	task-dependent	supervised, buffer of pairs matured after lag H	on label arrival	task-dependent

Table 2.2 expresses the principal families of adaptation methods as particular choices in Eq. 2.4. The remainder of this section discusses each family in turn.

2.4.3 Supervision regimes

Supervision determines the objective $\mathcal{L}$ and evidence buffer $\mathcal{B}_t$ in Eq. 2.4. Immediate ground truth, familiar from offline training, is unusual in deployed sensing. It requires an independent target measurement alongside every prediction. Delayed or absent targets lead to two further regimes. Under delayed supervision, a target arrives only after a horizon H , as in forecasting. The device must retain state for each prediction awaiting its label, so buffer occupancy grows with H and sampling rate even if no update follows. The eventual update also describes an earlier operating point. The adaptation loop contains a transport delay and corrects for conditions that have already passed. Without targets, adaptation uses an input-derived surrogate. Optimization-free methods replace normalization statistics to correct feature scale and offset; gradient-based methods optimize entropy or consistency among augmentations, assuming an adapted model is confident and stable [54, 55, 56, 57].

2.4.4 What is updated, and parameter-efficient adaptation

The trainable subspace $\mathcal{S}_t$ strongly affects update cost because backpropagation needs intermediate activations that inference can discard, together with optimizer state for trainable parameters. Peak memory during an update can therefore greatly exceed inference memory for the same model. This working-memory demand, rather than number of parameters alone, determines whether on-device learning is feasible [58, 59].

Head-only fine-tuning responds to this demand by freezing the encoder of Eq. 2.2. Backpropagation ends at the embedding, and encoder activations need not be retained. Other parameter-efficient schemes train inserted modules, low-rank factors, sparse weights, or normalization parameters [35]. They offer more capacity but require activation storage back to the shallowest trainable parameter.

2.4.5 When to update, triggers and gating

Of the three axes, the trigger receives the least explicit consideration, since much work assumes updating is always appropriate and reports what to update without asking whether to update at all. Drift-detection research addresses when to update [60]. Four signal families make different compromises. Scheduled triggers give predictable cost but ignore whether change is needed. Input-distribution triggers need no labels but detect changed inputs rather than deteriorating predictions. Uncertainty triggers use model confidence and inherit its miscalibration. Error triggers give direct evidence of performance loss but require supervision and react with a lag of H when labels are delayed.

Gating and triggering address different decisions, whether a sample appears suitable for learning and whether the available error evidence warrants an update, and neither guarantees the other. Evaluating their composition requires measurements of screening errors and their effect on subsequent predictions.

2.4.6 Failure modes of on-device adaptation

Adaptation introduces failure modes that static deployment does not have, and they share the property that the damage is written into the parameters and outlives the condition that caused it.

- **Catastrophic forgetting.** Sequential training on a non-stationary data stream reduces performance on earlier regimes, since the update rule includes no mechanism to retain skills that the current data no longer test. In deployment, the consequence is seasonal, as a model specialized to summer may have lost its winter behavior by the time winter returns. Established mitigation methods mostly require extra storage for replay samples, and so compete with budgets already committed on the device [6, 58].
- **Drift and divergence.** Even without forgetting, an update law driven by a noisy or biased signal can move parameters progressively away from a valid operating point, and under unsupervised surrogates the risk is acute, since a self-reinforcing objective admits no correction from measurement. Practical safeguards limit the step size without guaranteeing convergence, namely dead zones, gradient-norm clipping, small learning rates, and restriction of the reachable set through S_t .

- **Cost overrun.** Adaptation draws on the budget that inference already occupies, so an update policy that is algorithmically sound but temporally unbudgeted causes deadline misses in the pipeline it was meant to improve. The update rate a trigger implies usually limits the pipeline more than the cost of one update, so the trigger belongs in the feasibility assessment together with the update rule.

2.5 Collaboration across devices

A device can adapt only from what it has observed; with delayed supervision, even its error evidence arrives late. Across a fleet, another site may already have encountered the relevant regime. Distributed learning seeks to share that experience while keeping raw measurements local, in which bandwidth, intermittent connectivity, or data-governance constraints restrict central collection.

2.5.1 Federated averaging and its failure under structured heterogeneity

In federated learning (FL), clients train locally and periodically share model updates with a coordinator [61, 62]. Each round distributes a global state, lets clients train for a fixed number of epochs, and aggregates their returned states, commonly by local-sample-weighted averaging. Clients operate independently between rounds. Keeping observations local while sharing learned patterns can give a node access to experience of regimes it has not yet encountered.

In edge-device sensing applications, clients are physical installations whose data differ systematically by site, sensor, and operating regime. Appearance, target prevalence, and sample counts can all vary; sample-weighted aggregation also favors larger participants [63]. After several local epochs, clients may approach different optima whose average suits none of them. Collaboration can then perform worse for a client than isolated training, an outcome called negative transfer. Parameter averaging transmits both transferable structure and site-specific idiosyncrasies, and structured heterogeneity can allow the latter to dominate [62].

2.5.2 Design responses and the exchanged object

Responses to heterogeneity can be organized by which part of the loop they modify. A client's local training can be constrained so that it does not drift far within a round [64]; distribution-specific parameters such as normalization layers can be excluded from aggregation [65]; the model can be split so that a common representation is aggregated while the mapping to outputs stays individual, following the decomposition of Eq. 2.2 [66]; or each client can blend the aggregate into its own

state rather than replace it [67]. These approaches modify local objectives, aggregated parameter subsets, or aggregation dynamics. Their exchanged state therefore need not be identical, even though each stays a form of parameter exchange.

The transmitted object also determines cost and exposure. Full-state exchange scales with model size. Prototype exchange instead communicates aggregated embeddings associated with a class or regime [68, 69], so payload scales with embedding width and class count independently of encoder depth. This summary is narrower than a parameter state that describes behavior throughout the input space, but it is still a function of local data and may leak information [70]. Secure aggregation and differential privacy add formal protection at additional cost [71, 72]. The collaboration mechanisms examined in this thesis are stated in Sec. 2.6.2.

2.6 Positioning and the research gap

The deployment, adaptation, and collaboration literature supplies methods for different parts of the sensing lifecycle. The hypothesis in Sec. 1.2 connects task-specific pipelines, hardware-aware execution, selective updates, and compact exchange through their predictive and resource consequences. The contribution is a set of workload-specific implementations and evaluations organized using a shared framework. Table 1.1 maps each research question and objective to lifecycle stages and case studies.

2.6.1 Reliability criteria by lifecycle stage

Three groups of criteria, one per lifecycle stage, define how reliability is assessed alongside the performance and cost comparisons used to examine the hypothesis. Deployment concerns feasibility, physical consistency, and predictive robustness. Adaptation concerns input admissibility, update justification, and update affordability. Collaboration concerns predictive preservation, communication efficiency, and participation resilience. The evidence for a criterion takes the form of measurements for particular configurations, qualitative observations for some mechanisms, or untested requirements for others, and these statuses are retained when assessing the hypothesis (Sec. 2.7.1). Data locality is a cross-cutting design constraint, with raw observations retained at their originating clients. It does not establish a privacy guarantee or replace predictive preservation.

1. **Deployment feasibility.** The configuration runs on the target within its budgets while retaining the task performance that justified selecting it, which requires that parameters and peak activation memory fit the storage and random-access memory (RAM) available, that the model exports and compiles for the target runtime, and that end-to-end response time and energy comply with the deployment constraints (Secs. 3.2 to 3.4).

2. **Physical consistency.** The evidence a model relies on should be consistent with the physical mechanism, which is assessed by attribution against an independently derived description of the sensed process (Secs. 3.3 and 3.4).
3. **Predictive robustness.** Validated performance remains across the regime variability of the deployment without recourse to updating (Sec. 3.4).
4. **Input admissibility.** Screening should distinguish usable observations from sensing artifacts before learning, with missed corruptions and rejected valid observations both measured at a stated operating point, alone and in series with the update policy (Sec. 4.3).
5. **Update justification.** The realized evidence warrants a change, since updating on well-predicted samples expends energy and risks drift away from a still-valid operating point (Sec. 4.2).
6. **Update affordability.** Update latency, memory, and energy should fit the available device budget, measured on the execution path that performs the update (Sec. 4.4).
7. **Predictive preservation.** Collaboration should retain useful site-specific performance relative to isolated local adaptation, assessed by metric and site (Sec. 5.2).
8. **Communication efficiency.** Payload and synchronization cost should be assessed alongside predictive performance, including the incremental cost of peer exchange over local training (Sec. 5.2).
9. **Participation resilience.** Operation under missing or delayed clients requires explicit state-handling rules and a dropout evaluation that records participation per round (Sec. 5.2).

The encoder–head decomposition offers a shared interface for screening, head-only adaptation, and prototype exchange (Fig. 2.1). A frozen encoder can support inexpensive head updates, whereas the trainable encoder in Ch. 5 changes the embedding space and requires post-update prototype extraction.

2.6.2 Position taken in this thesis

The empirical chapters instantiate the mechanisms reviewed above as three component studies on a shared framework. In deployment (Ch. 3), models are exported, compiled for their targets and, where the toolchain permits, delegated to accelerators across the platforms of Table 2.6; physically informed inputs are evaluated by ablation on the forecasting workload, and attribution is checked against independently derived fault physics on the motor workload. In adaptation (Ch. 4), an error-triggered policy with a head-only and a reduced-step partial-encoder action is compared with static inference and with test-time, few-shot, and delayed-supervision alternatives on three recorded streams. An image-fault monitor is placed in series with it, so that exclusion is judged by its effect on

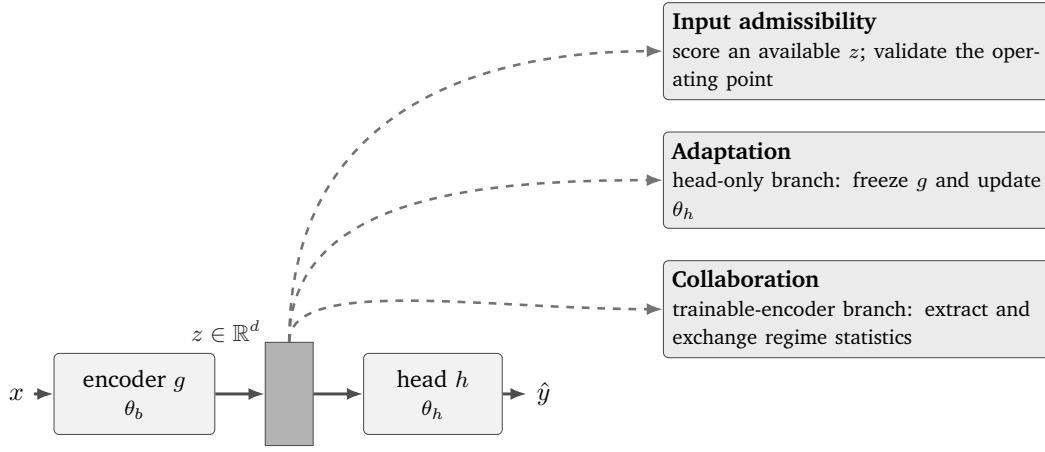


Fig. 2.1. The encoder–head interface for screening, head-only adaptation and prototype exchange. Dashed branches identify related uses, not one jointly evaluated execution path. Screening can reuse an exposed inference embedding; Ch. 5 additionally extracts prototypes after encoder updates.

learning and not by detection only. A frozen quantized encoder with a trainable host head is characterized on an inference accelerator with classification workloads. In collaboration (Ch. 5), regime-stratified prototype exchange with a trainable encoder and daily local training is compared with isolated local adaptation, with five parameter-exchange configurations, and with an own-prototype control, and its client cost is profiled on two boards. The encoder–head decomposition of Eq. 2.2 is the shared interface of the three studies (Fig. 2.1); their execution paths differ, and the studies are evaluated one stage at a time, which the assessment logic of Sec. 2.7.1 takes into account.

2.6.3 Research gap and mapping to the research questions

Table 2.3 delimits the thesis against its prior foundations. On-device training already addresses severe memory constraints [59, 35], federated averaging explicitly targets communication efficiency [61], and prototype exchange predates this dissertation [68, 69]. The questions here concern their application and evaluation in the selected sensing workloads.

The empirical gap concerns the conditions under which these mechanisms improve the selected sensing workloads. The deployment studies ask when task-specific visual structure refines prediction with a compact model, and when conversion or quantization enables on-device processing. The adaptation studies ask which conditional encoder actions avoid the observed deterioration, whether detecting an image fault justifies excluding it from supervised learning, and which update costs are actually measured. The collaboration study asks whether regime-level summaries for continuous regression add predictive value beyond local adaptation and own-

Tab. 2.3. Prior foundations, the open questions this evaluation addresses, and the scope of each stage’s contribution. The third column states the questions taken up here, not assumptions attributed to all prior work.

Stage	Prior foundations	Open evaluation question	Contribution and scope
Deployment Static on-device inference	Compression, quantization, and accelerator mapping make a fixed model executable under tight constraints [73, 22, 45, 43], and training itself has been fitted into microcontroller memory [59, 35]. Attribution methods expose what a model relies on [50, 51].	How do task-specific pipelines and hardware-aware execution affect predictive performance, complexity, and feasibility (RQ1 and RQ2)?	Conversion and execution measurements on selected platforms, qualitative attribution checks, and a physically informed input representation.
Adaptation Local post-deployment adaptation	Test-time adaptation restores predictive performance under shift without labels [54, 55, 56], continual variants guard against error accumulation on long streams [57], and delayed-supervision schemes update only the final layer or a rolling window [74, 75].	How do selective updates affect predictive outcomes and update cost, and how does execution change that cost (RQ2 and RQ3)?	Screening, lagged conditional-update and accelerator-execution studies, with screening evaluated in series with the policy and the accelerator path on its own workload.
Collaboration Cost-aware collaboration	Communication-efficient federated optimization and variants for statistical heterogeneity [61, 64, 65]; prototype methods exchange representation summaries [68, 69].	How does compact exchange compare with isolated adaptation and parameter exchange in predictive performance and collaboration cost (RQ4)?	A regime-stratified prototype mechanism for solar irradiance regression at preserved forecast skill and a site-dependent peer increment isolated by an own-prototype control.

prototype regularization at a smaller payload than full-state exchange. These are component comparisons, each having its own reference and measurement scope.

2.7 Experimental framework

This section defines the shared reporting framework through the sensing workloads and datasets, the reference platforms, the task-level metrics, the system-level costs and their measurement procedure, and the reporting conventions. Predictive performance and execution cost are considered together when assessing the hypothesis, with measurement coverage reported for each study. The empirical chapters share platforms, metric definitions, and measurement procedure; they do not share one matched dataset or configuration, and Sec. 2.7.1 states how the resulting evidence is interpreted.

2.7.1 Assessment logic

The central hypothesis is assessed through the component comparisons of Chs. 3 to 5 and through the coverage of the nine criteria. Each comparison uses the predictive metrics, reference methods, and cost measures defined for its task, and evidence of improvement takes one of two forms, better predictive performance with a smaller or cheaper model, or a lower measured execution or communication cost at comparable predictive performance. The number of parameters characterizes model complexity and does not stand in for measured memory or energy, and where a cost reduction comes with a predictive loss, the result is reported as a trade-off whose desirability depends on the application. The reference for each question is fixed in advance. RQ1 uses the reference pipelines and the representation ablations; RQ2 compares supported architecture, precision, runtime, and hardware settings, including the CPU reference for accelerator-backed training; RQ3 uses static inference and the adaptation policies; and RQ4 distinguishes isolated local adaptation, which exchanges nothing, from full-state parameter exchange, which sets the payload reference.

2.7.2 Sensing workloads and datasets

Three sensing workloads supply the evidence of this thesis, and Table 2.4 summarizes the corresponding datasets and the role each plays. The two thermal workloads enter at deployment only. In contrast, solar irradiance forecasting runs through all three lifecycle stages, because it is the only workload supplying continuous, delayed supervision across multiple sites. The physical and operational specifics of each workload are developed where it is studied.

Low-resolution occupancy estimation. A 32×24 far-infrared thermal array is mapped to a density map from which an occupancy count is derived, on the Thermo Presence dataset [76, 12]. The low resolution limits directly identifying visual detail

Tab. 2.4. Datasets and deployment streams used in this thesis, with the partition or split each chapter uses. Days are unique days with at least one usable entry after preprocessing. The all-sky streams pair imagery with co-located irradiance at a one-minute cadence; the week-scale streams are single-season collections. Coordinates are given for the sites of the multi-site campaigns.

Asset	Modality and labels	Period and coverage	Partition or split	Used in
Thermo Presence [76]	32×24 far-infrared array; density maps and derived counts	13,644 annotated frames	By recorded sequence (Table 3.2)	Sec. 3.2
Induction-motor thermography [13]	640×512 thermal at 4 FPS; three motor states	42 trials on one test stand	Five folds assigned in groups of frames, as the published fold sizes establish	Sec. 3.3
Folsom [77, 78], first partition	All-sky RGB and pyranometer readings, 1 min	38.642°N, 121.148°W; 2014 to 2016	Test 2014 (224,335 samples); training and validation 2015 to 2016 (495,371)	Sec. 3.4
Folsom, second partition	Same record	2016 held out; 363 days, 246,164 samples, of which 362 days and 245,538 samples enter the 365-day replay (29 February excluded)	Training 2014 to 2015, with 20 % of training days held out by whole days for validation	Secs. 4.2, 4.3 and 5.2
Poznań, campaign	All-sky RGB and pyranometer readings, 1 min	52.403°N, 16.954°E; 18 Oct 2024 to 1 Jul 2025; 213 days and 123,523 samples	Unseen site	Secs. 4.2 and 5.2
Karpicko, campaign	Same	52.129°N, 16.123°E; 1 Nov 2024 to 31 Oct 2025; 298 days and 179,858 samples	Unseen site	Secs. 4.2 and 5.2

and allows models to fit MCU-class devices, so the workload is the lower-limit reference on the feasibility spectrum, constraining models to information-limited sensing within kilobyte-scale SRAM budgets. Section 3.2 details the sensor, the dataset split, and the task formulation.

Non-invasive industrial fault diagnosis. High-resolution thermal imaging of squirrel-cage induction motors is mapped to an operating-state class [13, 14]. Because the thermal field is shaped by load, ambient conditions, and installation as well as by fault state, a sufficiently expressive model can learn the test stand rather than the fault, which makes this the physical consistency case study of the thesis. It serves both as an inference-feasibility benchmark across heterogeneous runtimes and accelerators and as the case in which attribution is checked against independently derived fault physics. Section 3.3 details the workload.

Intra-hour solar irradiance forecasting. A synchronized stream of all-sky red, green, blue (RGB) images and recent irradiance history is mapped to a forecast at horizon $H \in \{5, 10, 15, 30\}$ minutes. The irradiance history enters at the embedding, without an encoder of its own, so the forecaster retains the single-encoder cost structure

of Eq. 2.2. The base model is trained on the Folsom dataset [77, 78], and three deployment subsets supply cross-site geographic and seasonal shift [15]. The three deployment sites and the two Folsom partitions are listed in Table 2.4. The cameras, lenses, and exposure control differ between the sites (Fig. 4.2), so the deployment streams differ from the training distribution in instrumentation as well as in site and season. Because supervision is delayed by the forecast horizon, the node must buffer predictions and the minimal update state until the ground truth arrives, which makes this workload the testbed for on-device adaptation (Ch. 4) and cross-site collaboration (Ch. 5).

The two Folsom partitions and the two forecasters. The Folsom record has two independent partitions. Chapter 3 tests on 2014 after training and validation on 2015–2016. Chapters 4 and 5 instead train on 2014–2015 and replay 2016 as an unseen stream.

The first partition adopts the split of the edge-oriented comparator of Papatheofanous et al. [79], so that the reimplemented forecasters of Table 3.11 are scored on the test year of their own publication. The second partition is chronological. Adaptation and exchange advance through the replayed stream in time, and a checkpoint fitted on later years would carry information from the future of that stream into every update.

The years differ in cloud climatology, so their absolute errors are not comparable even at matching horizons. The later studies inherit the encoder architecture, the encoder–head factorization, the loss, and the pretraining recipe. They do not inherit the fitted weights or the input channels. A physics-informed forecaster was also trained on the second partition for the crossed checks of Chs. 4 and 5. Table 2.5 compares the configurations.

2.7.3 Reference edge platforms

The empirical results of Chs. 3 to 5 are based on a fixed set of execution environments spanning the four device classes of Table 2.1, with at least one instance per class so that the four regimes reappear throughout the thesis. Table 2.6 lists the instances and the role each plays; the measurement procedure of Sec. 2.7.5 standardizes how their constraints are characterized and reported.

2.7.4 Task families and task-level metrics

On-device perception workloads in this thesis are structured around four families of predictive tasks, which differ in the form of the output.

- **Classification.** Assignment of discrete labels or system states to individual frames or short segments.

Tab. 2.5. Forecaster configurations. The deployed configuration is one checkpoint shared by Chs. 4 and 5; its loss, learning rate, and schedule are those stored in the checkpoint. The physics-informed forecaster of the crossed checks in Chs. 4 and 5 uses the deployed column’s partition, loss, and optimizer with the seven-channel input. Post-deployment settings differ between the chapters.

	IEP forecaster Sec. 3.4	Deployed configuration Secs. 4.2, 4.3 and 5.2
Input	Seven-channel IEP tensor, 128×128 , plus $I_t, I_{t-5}, I_{t-10}, I_{t-15}$	RGB frame, 128×128 , plus the same four values; the seven-channel IEP tensor in the crossed checks of Figs. 4.6 and 5.5
Encoder	ResNet50, ImageNet-pretrained	ResNet50, ImageNet-pretrained
Head	Linear (2052), ReLU, Linear (256)	Linear (2052), ReLU, Linear (256)
Loss	BerHu	BerHu; MSE for the online update step
Folsom partition	Test 2014; training and validation 2015 to 2016	Test 2016; training and validation 2014 to 2015
Optimizer	AdamW, 3×10^{-4}	AdamW, 3×10^{-4}
Batch, epochs, stopping	128 (96 and 64 at the two largest resolutions); learning rate halved after five epochs without validation improvement	64; early stopping after five epochs without validation improvement; learning rate halved after two
Online settings (Ch. 4)	None, static	One step per buffered sample; weight decay 10^{-4} ; moment decay 0.5 and 0.975; clipping at norm 1.0; $T_1 = 10 \text{ W/m}^2$, $T_2 = 100 \text{ W/m}^2$; head learning rate 10^{-5} , and for the proposed instantiation the last two encoder stages at 10^{-6}
Client settings (Ch. 5)	None	One local epoch per day; AdamW 10^{-5} ; weight decay 10^{-4} ; clipping at norm 1.0; batch 32; replay of the 1,000 most recent samples at ratio 0.5; no warm-up; synchronization every 7 days
Checkpoint	Archived artifact (Sec. 6.7)	One checkpoint shared by Chs. 4 and 5; a second, physics-informed checkpoint on the same partition for the crossed checks of Figs. 4.6 and 5.5 (Sec. 6.7)

- **Dense prediction and counting.** Estimation of a spatially resolved output, most commonly a density map whose integral yields a scalar count. It is preferred to direct classification of the scalar wherever that quantity is unbounded, or its range is unknown at design time (Sec. 3.2).
- **Regression and forecasting.** Estimation of continuous variables from current context and, where applicable, prediction of their evolution over a horizon H .
- **Anomaly detection.** Identification of out-of-distribution events, faults, or corruptions relative to a nominal regime. A detector is evaluated by its detection rate on corrupted inputs, its flag rate on nominal inputs, and the resulting coverage, together with its effects on learning and execution cost. A verified false-alarm rate additionally requires fault-free labels.

The metrics of the three families evaluated on their output are counting mean absolute error (MAE), MSE and mean absolute percentage error (MAPE) with dense-

Tab. 2.6. Reference edge platforms employed in the dissertation and their function in the experimental design. The key envelope indicates the dominant resource regime (a qualitative, configuration-dependent characterization) used to interpret latency, memory, and energy measurements. Raspberry Pi memory configurations vary across studies. The AI accelerators were paired with Raspberry Pi 4B or Raspberry Pi 5, depending on configuration.

Platform	Clock and memory	Runtime and precision	Role
<i>MCUs (bare-metal)</i>			
Arduino Nano 33 BLE Sense (nRF52840)	64 MHz; 1 MB / 256 kB	TFLM; INT8, FP32	Tightest memory envelope in the dissertation; sets the lower feasibility limit under static memory planning.
Arduino Portenta H7 (STM32H747)	480 MHz; 2 MB / 1 MB	TFLM; INT8, FP32	Same runtime as above at higher clock rate, isolating clock from runtime effects.
STM32 F429ZI Nucleo-144	180 MHz; 2 MB / 256 kB	X-CUBE-AI; INT8, FP32	Vendor toolchain against the open runtime at comparable memory.
STM32 H745ZI Nucleo-144	480 MHz; 2 MB / 1 MB	X-CUBE-AI; INT8, FP32	Isolates toolchain from silicon, since the Portenta H7 shares the same core family.
LOLIN32 (ESP32-WROOM-32)	240 MHz; 4 MB / 520 kB	TFLM with ESP-NN; INT8, FP32	Shows whether a vendor kernel library changes the accuracy-cost trade at fixed precision.
<i>SBCs (CPU-only)</i>			
Raspberry Pi 4B [80]	1.5 GHz Cortex-A72; 2 GB–8 GB	TensorFlow Lite, ORT, PyArmNN; FP32/FP16, INT8 (toolchain-dependent)	Baseline SBC; shows the limits of CPU-only inference under tight power and thermal envelopes.
Raspberry Pi 5 [81]	2.4 GHz Cortex-A76; 4 GB; single-digit W; thermally limited	PyTorch, ORT; FP32/FP16 (where supported), INT8 (select ops)	CPU-only baseline of the later chapters, where inference and updates share one CPU.
<i>AI accelerators</i>			
Coral USB (Edge TPU)	500 MHz; 8 MB on-chip SRAM	TensorFlow Lite; INT8 on accelerator, FP32/FP16 on host	Inference-centric offload; only supported subgraphs accelerate, so host work must be accounted for.
Intel NCS2 [82]	700 MHz Myriad X VPU; host DRAM	OpenVINO, ORT; FP32/FP16, no INT8 on-device	Contrasting accelerator type; no integer support and a toolchain that limits what can be offloaded.
Hailo-8L [83]	Host DRAM + INT8 NPU; accelerator memory is constrained; shared power/thermal budget	INT8 on accelerator; FP32 on host	Partition used for adaptation; a frozen encoder is offloaded while trainable and unsupported parts stay on the CPU.
<i>Embedded GPU</i>			
Jetson Orin Nano [84]	Multi-GB DRAM/VRAM; configurable power modes from 7 W to 40 W, MAXN with <code>jetson_clocks</code> enabled in this dissertation; high bandwidth	FP16/FP32, INT8 (toolchain-dependent)	Upper baseline for throughput and for the energy cost of the heterogeneous design.

map MAE and MSE for dense regression (Ch. 3), accuracy, precision, and F1 score for classification (Ch. 3), and MAE, root mean squared error (RMSE), MAPE, normalized root mean squared error (nRMSE) and forecast skill (FS) for time-series regression and forecasting (Chs. 3 to 5). For dense outputs, n denotes the number of evaluated samples and $H \times W$ the spatial resolution; for streaming data, $t \in \{1, \dots, n\}$ denotes discrete time steps, and y_t and $\hat{y}_t$ the measured and predicted values; a small constant $\epsilon > 0$ is added in denominators to prevent division by zero. Accuracy, precision, F1

score, MAE, MSE, RMSE and MAPE follow their standard definitions and are not restated. For multiclass experiments, per-class precision and F1 score are computed one-vs-rest and aggregated by class support.

Dense regression and counting. For a density-map target, MAE and MSE are computed element-wise and averaged over the $n \cdot H \cdot W$ elements of the evaluation set. The count $\hat{c}_h$ derived from a predicted map (Sec. 3.2) is scored against the annotated count l_h with

$$\text{MAE}_{\text{count}} = \frac{1}{n} \sum_{h=1}^n |l_h - \hat{c}_h|, \quad (2.5)$$

$$\text{MSE}_{\text{count}} = \frac{1}{n} \sum_{h=1}^n (l_h - \hat{c}_h)^2. \quad (2.6)$$

Mean absolute percentage error on counts is defined as

$$\text{MAPE}_{\text{count}} = \frac{100\%}{n} \sum_{h=1}^n \frac{|l_h - \hat{c}_h|}{\max(l_h, 1)}, \quad (2.7)$$

where the denominator is clamped at one so that empty frames remain scored. Empty frames carry no annotated occupant, so a percentage error is undefined for them in the usual sense; clamping assigns them the absolute error instead, which is zero whenever such a frame is counted correctly.

Time-series regression and forecasting. For time-series regression, including very-short-term solar irradiance forecasting, MAE, MAPE, and RMSE are reported over a stream of length n , together with the nRMSE

$$\text{nRMSE} = \frac{\text{RMSE}}{\bar{y}} \cdot 100\%, \quad (2.8)$$

where $\bar{y}$ is the mean of y_t over the evaluation set. In solar forecasting, MAPE is interpreted with care at very low irradiance, where the denominator dominates percentage errors.

To quantify improvement over a domain baseline, FS is additionally reported,

$$\text{FS} = \left(1 - \frac{\text{RMSE}_M}{\text{RMSE}_P}\right) \cdot 100\%, \quad (2.9)$$

where RMSE_M and RMSE_P denote the RMSE of the model and of the baseline. FS is reported wherever a domain baseline is evaluated alongside the model and is the primary summary of forecasting performance within each study. Because the Folsom partitions and the deployment streams differ between chapters (Sec. 2.7.2), forecast skill values from different chapters are reported side by side and are not treated as one matched comparison; within a chapter, comparisons hold at a fixed site, horizon,

and sample mask. The baseline used throughout is Smart Persistence, which persists the clear-sky index rather than the raw irradiance,

$$\hat{I}_{t+H} = \left(\frac{I_t}{I_{cs,t}} \right) \cdot I_{cs,t+H}, \quad (2.10)$$

where $I_{cs,t}$ denotes clear-sky irradiance from the Ineichen–Perez model at the site location [85]. Persisting the clear-sky index accounts for the deterministic diurnal component. The regime labels of Sec. 5.2.1 use the same clear-sky normalization, with cut points 0.30 and 0.85 on the clear-sky index crossed with four clear-sky irradiance levels.

2.7.5 System-level metrics and measurement procedure

Feasibility depends on predictive performance and the operating costs of obtaining it. Each study reports the subset of latency, throughput, memory, and energy its setup covers; comparison requires coordinating the model, runtime, numerical format, input, batch size, measurement scope, power mode, and thread settings. The following definitions apply throughout.

- **Latency (ms).** Steady-state wall-clock time per sample unless stated otherwise. Sensor readout and the acquisition buffer are excluded, as Eq. 2.1 defines; the memory copies into the input tensor count in T_{pre} , and host–accelerator transfers count in T_{infer} . Model-only latency is T_{infer} and end-to-end latency is T_{e2e} . Both are given, since a model that fits the budget on its own may still miss the end-to-end requirement.
- **Throughput (sample/s).** Stable processing rate under continuous streaming input. Unless stated otherwise, throughput is measured at batch size 1 to approximate sensor-loop operation.
- **Units and cost measures.** Memory, storage, and payload sizes use decimal kB, MB and GB throughout; microcontroller SRAM and flash capacities retain the convention of the cited data sheet. Compute cost is given either as FLOPs, the number of floating-point operations for one forward pass at the stated input resolution, or as multiply–accumulate operations (MACs), where one MAC corresponds to two FLOPs; each table states which of the two it uses.
- **Peak memory (MB).** Peak resident RAM during execution is distinct from model-file size and parameter storage. Update-time memory includes retained activations and optimizer state. Swap was disabled on the Raspberry Pi 5 and the Jetson Orin Nano, so every reported peak was held in physical memory.
- **Energy (J) and power (W).** In multi-device studies, energy is measured mainly with external inline power meters, which reduce platform-specific bias. On the Jetson platform, platform-wide telemetry is collected using `tegrastats`, while

on the Raspberry Pi 5 the power-management integrated circuit is queried via `vcgencmd pmic_read_adc`.

Feasibility is claimed per measured quantity, and neither low inference latency by itself nor a small transmitted payload is taken as evidence of continuous operation within the full memory, energy, and deadline budget.

2.7.6 Reporting conventions

Each metric is computed over the stated evaluation stream at each site, with a common sample mask per metric and a stated temporal aggregation rule; cross-site summaries are unweighted means of site metrics, and Chs. 4 and 5 pool MAE and RMSE over the population, with the Smart Persistence RMSE evaluated on the same population for forecast skill. FS is the principal summary of a forecasting comparison within a study; MAE describes the typical error, and RMSE weights the large ones, and a gain in one is always reported with the other two. Adaptation is compared with both the frozen model and the best alternative the device could run, together with its update cost, since adaptation draws on the inference budget. Replication differs by study and is stated with each result. The deployment studies use one training run per configuration, except the five-fold motor evaluation; the adaptation comparison uses one run; and the collaboration comparison uses five seeds for the proposed and isolated arms and three for each parameter-exchange configuration.

Reliable sensing and inference on the edge

Moving deep neural networks onto embedded hardware facilitates real-time, on-device inference and local control, but gives up the scalable compute and centralized monitoring of the cloud. This chapter addresses RQ1 and the deployment part of RQ2 in Sec. 1.2 through the deployment criteria of Sec. 2.6.1. Its three visual-sensing studies span hardware from microcontrollers with kilobytes of SRAM to single-board computers and AI accelerators, and each stresses a different constraint. Thermal occupancy counting (Sec. 3.2) shows that a density-regression pipeline halves the counting error of its reference with $3.55\times$ fewer parameters and runs from integer artifacts of 59 kB to 182 kB on MCU-class boards. Induction motor diagnosis (Sec. 3.3) shows that the conversion toolchain, not cross-validated accuracy, decides which architectures can be deployed, and checks what the classifier attends to against the physics of each fault. All-sky irradiance forecasting (Sec. 3.4) quantifies the gain from physics-informed input channels within a fixed backbone family and locates the limit of that gain at an unseen site. Together, these findings develop the task-specific representation contribution C2 and the hardware-aware execution contribution C4 of Sec. 6.2. These three studies build on the author's publications [11, 13, 15] and on two related contributions [12, 14].

3.1 Deployment criteria and evaluation protocol

Moving inference onto the device changes how failure behaves. An unsupported operator blocks compilation, an activation spike overflows SRAM, and an exceeded latency limit invalidates a control cycle. These are hard failures that occur before any output exists, and a model that appears robust on a static IID test set adds a second class of failure once the deployment drifts [6, 86]. The constraints separated in Sec. 2.1.2 must therefore be satisfied simultaneously on one platform, and Sec. 2.6.1 groups deployment reliability into feasibility, physical consistency, and predictive robustness, assessed on pre-installation data. Deployment feasibility, item 1 of Sec. 2.6.1, is applied here through its three clauses, memory feasibility (parameters and peak activations fit the nonvolatile storage and on-chip RAM, with no virtual memory to absorb overflow), toolchain support (export, conversion, and compilation with operators and numerical formats admitted by the target runtime), and budget compliance (end-to-end response time and energy within the response-time and duty-cycle limits of the installation), each while retaining the

Tab. 3.1. Evaluation protocol of the three deployment studies. Task metrics are defined in Sec. 2.7.4, system metrics follow Sec. 2.7.5, and the platforms are those of Table 2.6.

Study	Constraint and criterion	Platforms and runtimes	Task metrics	System metrics	Replication
Occupancy counting (Sec. 3.2)	Memory; deployment feasibility	Five MCU boards with TFLM and X-CUBE-AI; Raspberry Pi 4B with TensorFlow Lite; Coral Edge TPU; Intel NCS2	Dense-map MAE and MSE; counting MAE, MSE and MAPE; accuracy and F1 score on count bins	Inference latency; board-level power in the inference and low-power states; duty-cycle energy	One training run; repeated latency
Motor diagnosis (Sec. 3.3)	Toolchain; deployment feasibility and physical consistency	Raspberry Pi 4B with TensorFlow Lite, PyArmNN and ORT; Intel NCS2 with ORT and OpenVINO	Accuracy, precision and F1 score, weighted by class support, under five-fold cross-validation	Conversion outcome per runtime and precision; per-frame latency	One model per fold; repeated latency
Irradiance forecasting (Sec. 3.4)	Time budget; deployment feasibility and predictive robustness	Raspberry Pi 4B and Raspberry Pi 5, CPU inference	MAE, MAPE, RMSE, nRMSE and FS against Smart Persistence	Stage-wise latency against the acquisition period	One training run; repeated latency

task performance that justified the configuration. Because a change that satisfies one clause can violate another, the unit of assessment is the complete configuration of task, architecture, precision, runtime, and platform [73]. Section 3.2 stresses memory, Sec. 3.3 toolchain support together with physical consistency, and Sec. 3.4 the time budget together with predictive robustness.

Table 3.1 summarizes the platforms, runtimes, metrics, and replication of the three studies. Latency is measured repeatedly and reported as a mean with its standard deviation. In contrast, task metrics come from one training run per configuration, except in the motor study, which trains one model per cross-validation fold. One quantity recurs across the studies. Because the installations sample periodically, with period T_{cycle} , the active fraction $T_{\text{e2e}}/T_{\text{cycle}}$ states how much of each period the processor spends on the pipeline. Section 3.2 uses it to derive daily energy, and Sec. 3.4 to estimate the compute that remains available for other work, which Ch. 4 later uses for adaptation.

3.2 Deployment feasibility under kilobyte memory budgets

On MCU-class hardware, the question is not how accurate a model can be made, but whether any useful model fits at all, so deployability rather than task

Tab. 3.2. Thermo Presence dataset [76] split used for training, validation, and testing. Counts are frames per number of people; sequences were assigned whole to the subsets (Sec. 3.2.1).

Split	Number of people in frame						Total
	0	1	2	3	4	5	
Training	99	105	2,984	3,217	1,953	114	8,472
Validation	0	139	631	1,691	225	139	2,825
Test	162	83	211	341	1,235	315	2,347

performance is the quantity to establish. This section stresses the memory clause of Sec. 3.1 on the class of target where it is most severe, using thermal people counting on a 32×24 far-infrared array, the lower-limit reference of the feasibility spectrum introduced in Sec. 2.7.2. Occupancy counts inform HVAC start and closure schedules and ventilation control [9], and a low-resolution thermal array suits this use because it reveals little identifying visual detail and produces small tensors. The same resolution removes the edges and textures that vision architectures exploit, so the task is reshaped to stay well posed on weak evidence. A density-regression formulation, an operator-simple architecture, and post-training INT8 quantization employing the target toolchain together yield a pipeline that fits, compiles, and runs on boards with hundreds of kilobytes of RAM while holding counting error within the annotated occupancy range. The study considers the counting and device-execution components of such a sensor; closed-loop building control is outside its scope.

3.2.1 Method

Dataset. The Thermo Presence dataset [76] (Table 2.4), recorded with the MLX90640 far-infrared sensor [87] in several indoor environments and room layouts, is used with the training, validation, and test split of Table 3.2, illustrated in Fig. 3.1. The split assigns whole recorded sequences to subsets, so no frame of a sequence appears in two subsets, and it is not stratified by occupancy. Four-person frames make up 23 % of the training set and 53 % of the test set, five-person frames 1.3 % and 13.4 %, and the validation set contains no empty frames. Hence, the test set is harder than the training distribution at the upper end of the occupancy range.

Task formulation. Occupancy could be modeled as a multi-class classification over the count set $\{0, 1, \dots, K\}$, but under limited observability this formulation is fragile. A person may cover only a few pixels, and each decision then rests on very weak evidence. Moreover, the fixed label space caps the output at the largest count seen in training, which turns a sensing constraint into a system limit, as in the system of Metwaly et al. [88]. The model here instead estimates a dense occupancy map from which the count is derived. Following Kraft et al. [76], each annotated person center

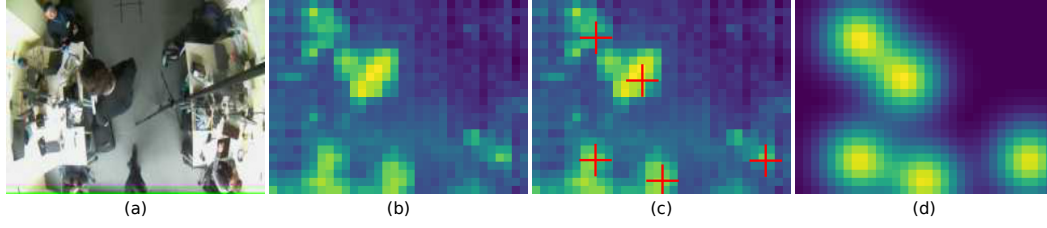


Fig. 3.1. Example from the Thermo Presence dataset: (a) corresponding and illustrative standard color image; (b) thermal image; (c) thermal image with annotated person centers; (d) the obtained density map.

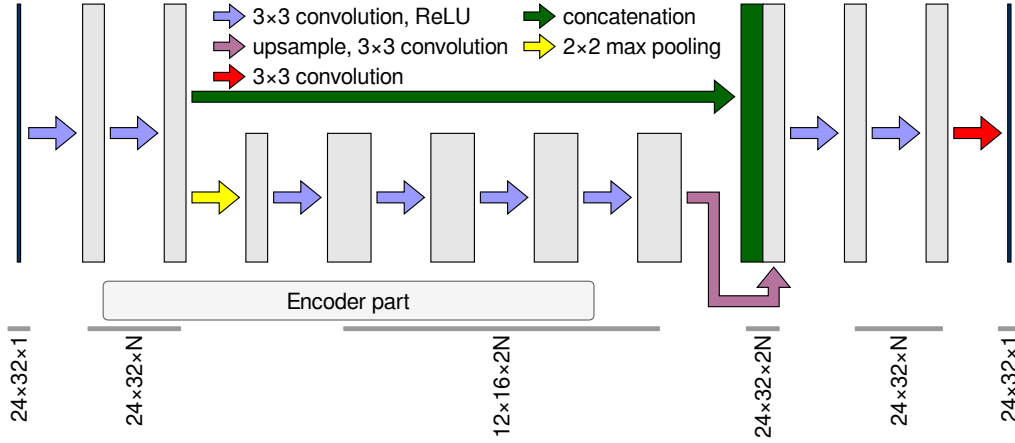


Fig. 3.2. A lightweight, computation-efficient U-Net variant employed for density map regression for occupancy estimation in low-resolution thermal images (32×24 frames). N denotes the number of convolutional filters, and in the final configuration it was set to 16.

becomes a peak in a sparse map that is convolved with a Gaussian kernel of standard deviation $\sigma = 3$ pixels, giving the ground-truth density map of Fig. 3.1. Because the mass of a component varies slightly across the field of view, for instance near the image edges, a global calibration factor α is computed from the training set,

$$\alpha = \frac{\sum_{h=1}^n \sum_{i=1}^H \sum_{j=1}^W y_{hij}}{\sum_{h=1}^n l_h}, \quad (3.1)$$

where y_{hij} is the ground-truth density at pixel (i, j) of training example h , l_h is its annotated count, n is the number of training examples, and $H \times W = 24 \times 32$ is the frame size. At inference, the count follows from the predicted map $\hat{y}$ as

$$\hat{c}_h = \text{round} \left(\frac{\sum_{i=1}^H \sum_{j=1}^W \hat{y}_{hij}}{\alpha} \right), \quad (3.2)$$

and this integer count is scored with the counting metrics of Sec. 2.7.4, with the percentage error of Eq. 2.7 averaged over all test frames, empty frames included.

Architecture and training. A spatially resolved output requires an encoder–decoder design, in which a downsampling path reduces spatial size while increasing channel

depth, an expanding path restores the resolution, and skip connections carry fine detail across the bottleneck. The U-Net family [89] is the standard example, and suits constrained deployment because its depth, channel width, and number of resolution levels can be reduced substantially while the topology and operator set stay unchanged. The operator set matters as much as the size, since a model built from standard convolutions and pooling compiles without changes across embedded runtimes, whereas a less common primitive can block conversion entirely. The single-channel, strongly reduced U-Net of Fig. 3.2 therefore uses only 3×3 convolutions employing ReLU activations, 2×2 max pooling, upsampling, and concatenation, with $N = 16$ filters in the first block and 46,577 parameters in total. It was trained in TensorFlow with the Adam optimizer at a learning rate of 10^{-3} , an MAE loss on the density map, a batch size of 128, and early stopping on the validation loss, which ended training after 50 epochs [11].

Deployment and measurement. The trained model is exported following the reduction step of Sec. 2.2.3. Post-training INT8 quantization within one converter reduces the TensorFlow Lite file from 192 kB at FP32 to 59 kB, which is also the size of the TFLM artifact, whereas X-CUBE-AI produces 383 kB and 182 kB at the two precisions, so the toolchain determines which artifact fits. Edge TPU INT8 and OpenVINO FP16 exports serve the accelerator configurations. The benchmark covers the five MCU-class boards, the SBC baseline, and the two accelerator-backed configurations of Table 2.6, executed with tflite-runtime 2.11.0, X-CUBE-AI, and the Edge TPU and OpenVINO delegates. Latency comes from the built-in timers of each platform, and the inference state includes receiving the input frame and transmitting the output map over the serial port. Power is board-level average power at one common supply voltage, logged with a Mooshimeter at 9 Hz, so it covers the whole development board, including on-board regulators, debug interfaces, and indicator LEDs. The low-power state is sleep mode on the microcontrollers and a power-optimized configuration on the Raspberry Pi, which offers no selectable sleep state. Because a 9 Hz logger cannot resolve a single inference, inference power is the mean over an interval of back-to-back inferences. In HVAC-style monitoring, inference runs periodically, so the energy per cycle is

$$W_{\text{cycle}} = T_{\text{infer}} \cdot P_{\text{infer}} + (T_{\text{cycle}} - T_{\text{infer}}) \cdot P_{\text{low}}, \quad (3.3)$$

where T_{cycle} is the sampling period, T_{infer} the per-inference latency, P_{infer} the average power during inference, and P_{low} the low-power draw, and daily energy follows as

$$W_{\text{daily}} = d_{\text{cycles}} \cdot W_{\text{cycle}}, \quad (3.4)$$

with d_{cycles} cycles per day. Sampling periods of 30 s and 5 min are evaluated.

Tab. 3.3. Prediction quality and model complexity against prior thermal people-counting baselines [11]. Dense-map metrics (MAE/MSE) measure pixel-level density fidelity, while count metrics quantify the inferred occupancy estimate (Eqs. 2.5 to 2.7). Both baselines were reimplemented and retrained on the identical dataset splits. Entries marked * are undefined for Metwaly et al. [88], which predicts a count directly and produces no density map; for that baseline the two-hidden-layer variant (512 units each) is reported, because the published single-hidden-layer model performed poorly on this more varied dataset.

Metric	Proposed U-Net	Kraft et al. [76]	Metwaly et al. [88]
MAE ↓	0.106	0.128	*
MSE ↓	0.033	0.051	*
MAE _{count} ↓	0.023	0.046	0.316
MSE _{count} ↓	0.023	0.049	0.458
MAPE _{count} ↓ (%)	0.81	1.70	21.03
Accuracy ↑ (count bins)	0.978	0.956	0.755
F1 score ↑ (count bins)	0.978	0.959	0.724
#parameters ↓	46,577	165,441	659,462

Tab. 3.4. Per-occupancy breakdown of counting performance on the held-out test split for the full-precision model of Table 3.3. Accuracy is the share of frames of each level counted exactly, MAE is the mean absolute counting error over frames of that level, and the last column gives the number of misclassified frames with the level assigned to them.

#of people in frame	Test frames	accuracy ↑	MAE ↓	Misclassified as (frames)
0	162	1.000	0.000	—
1	83	0.940	0.060	0 (3), 2 (2)
2	211	0.972	0.033	1 (5), 4 (1)
3	341	0.962	0.038	2 (13)
4	1,235	0.990	0.010	3 (6), 5 (6)
5	315	0.949	0.051	4 (15), 6 (1)

3.2.2 Evaluation

Counting performance. Compared with the retrained baseline of Kraft et al. [76] in Table 3.3, the proposed pipeline halves counting MAE from 0.046 to 0.023 with about $3.55\times$ fewer parameters, and its margin over the classification-style system of Metwaly et al. [88] is an order of magnitude in counting error at $14\times$ fewer parameters. Dense-map fidelity improves alongside the count, so the gain is visible before rounding. Because four-person frames dominate the test split, Table 3.4 breaks the result down by occupancy. Accuracy ranges from 94.0% for single occupancy to 100% for empty frames, and 51 of the 52 misclassified frames are confusions with an adjacent level. The error is small across the annotated levels, including the five-person level, which is ten times rarer in training than in test.

Task performance after export. The counting advantage remains after export on every configuration. Each row of Table 3.5 holds counting MAE at or below 0.038

Tab. 3.5. Task-level accuracy of the deployed density-regression model across MCU-class, SBC-class and accelerator-backed configurations [11]. Counting metrics use the same held-out split for every row, so differences indicate numerical precision and runtime kernels, since the model is the same. Because counts are rounded to integers (Eq. 3.2), counting MAE and MSE coincide wherever every error has magnitude one and separate slightly wherever an error of two occurs. Best value per column in bold. The platform cost of the same configurations is given in Table 3.6.

Device	Data type	MAE _{count} ↓	MSE _{count} ↓	MAPE _{count} ↓ (%)
<i>MCU-class</i>				
Arduino Nano 33 BLE Sense	INT8	0.023	0.024	0.82
Arduino Portenta H7	INT8	0.023	0.024	0.82
STM32 F429ZI Nucleo-144	INT8	0.036	0.038	1.14
STM32 H745ZI Nucleo-144	FP32	0.023	0.023	0.81
STM32 H745ZI Nucleo-144	INT8	0.036	0.038	1.14
LOLIN32	INT8	0.023	0.023	0.81
<i>SBC-class (CPU-only)</i>				
Raspberry Pi 4B	FP32	0.023	0.023	0.81
Raspberry Pi 4B	FP16	0.023	0.023	0.81
Raspberry Pi 4B	INT8	0.038	0.039	1.18
<i>Heterogeneous (SBC + accelerator)</i>				
Raspberry Pi 4B + Coral USB Accelerator	INT8	0.037	0.038	1.16
Raspberry Pi 4B + Intel NCS2	FP32	0.028	0.029	0.92
Raspberry Pi 4B + Intel NCS2	FP16	0.028	0.029	0.91

and counting MAPE at or below 1.18 %, so the weakest evaluated runtime still leads the strongest full-precision baseline of Table 3.3. The TFLM configurations reproduce the full-precision counting MAE within rounding. In contrast, the same INT8 model raises it by 0.013 to 0.015 on the X-CUBE-AI and Raspberry Pi paths, so the effect of integer export depends on the runtime, not on the nominal precision alone. Successful execution on the Arduino Nano 33 BLE Sense and the STM32 F429ZI, the boards with 256 kB of RAM, shows that the integer artifacts, 59 kB on the Nano and 182 kB on the F429ZI, and their activations fit the tightest memory envelope of the benchmark. These measurements satisfy the memory and toolchain clauses of Sec. 3.1 while retaining the task performance that justified the pipeline.

Energy under duty-cycled operation. At a 30 s sampling period the LOLIN32 (ESP32-WROOM-32) uses about 0.92 J per cycle and 2.64 kJ per day; at 5 min it uses 8.62 J per cycle and 2.48 kJ per day. Its daily consumption is about $2.7\times$ lower than that of the next-best microcontroller, the Arduino Nano 33 BLE Sense, and about $21\times$ lower than that of the fastest MCU-class target, the STM32 H745ZI. Energy per inference ranks the boards differently. The H745ZI at INT8 needs 60.5 mJ per inference compared with 85.7 mJ for the LOLIN32 (Table 3.6), but over a day the order reverses, because inference accounts for less than 10 % of daily energy on

Tab. 3.6. Platform cost of the configurations in Table 3.5. Latency and power are measured [11]; the daily-energy decomposition is derived from them. Latency is the mean over repeated inferences, with its standard deviation. Power is board-level (Sec. 3.2.1). Accelerator rows measure host and coprocessor together, and one power value per device serves all precision rows on the STM32 H745ZI, the Raspberry Pi 4B, and the accelerators. Hence, energy differences between precisions on these rows come from latency alone. E_{infer} is energy per inference, $T_{\text{infer}}P_{\text{infer}}$. Daily energy follows from Eqs. 3.3 and 3.4 at a 30-second duty cycle, and the last column gives the share of it spent on inference, the remainder being low-power-state draw. Lower is better for all numeric columns. Best value per column in bold.

Device	Data type	Avg. latency (ms)	P_{infer} (mW)	P_{low} (mW)	E_{infer} (mJ)	Daily energy (kJ)	Inference share (%)
<i>MCU-class</i>							
Arduino Nano 33 BLE Sense	INT8	$1,430.125 \pm 1.143$	116.62	79.88	166.8	7.05	6.8
Arduino Portenta H7	INT8	137.494 ± 0.500	881.08	514.75	121.1	44.62	0.8
STM32 F429ZI Nucleo-144	INT8	230.939 ± 0.100	740.31	517.81	171.0	44.89	1.1
STM32 H745ZI Nucleo-144	FP32	165.983 ± 0.100	1,136.79	638.66	188.7	55.42	1.0
STM32 H745ZI Nucleo-144	INT8	53.260 ± 0.100	1,136.79	638.66	60.5	55.26	0.3
LOLIN32	INT8	840.442 ± 0.021	101.93	28.53	85.7	2.64	9.3
<i>SBC-class (CPU-only)</i>							
Raspberry Pi 4B	FP32	7.707 ± 0.591	2,994.96	1,798.76	23.1	155.44	<01
Raspberry Pi 4B	FP16	7.690 ± 0.558	2,994.96	1,798.76	23.0	155.44	<01
Raspberry Pi 4B	INT8	4.194 ± 0.052	2,994.96	1,798.76	12.6	155.43	<01
<i>Heterogeneous (SBC + accelerator)</i>							
Raspberry Pi 4B + Coral	INT8	0.570 ± 0.060	4,176.18	3,223.16	2.4	278.48	<01
Raspberry Pi 4B + Intel NCS2	FP32	2.630 ± 0.159	4,447.74	2,479.11	11.7	214.21	<01
Raspberry Pi 4B + Intel NCS2	FP16	2.300 ± 0.100	4,447.74	2,479.11	10.2	214.21	<01

all platforms and less than 1 % on most. The remaining time is charged at the low-power draw, which is 28.5 mW for the LOLIN32 and 639 mW for the H745ZI development board.

The inference share in Table 3.6 therefore describes how daily energy splits between the two states. It is not a measure of efficiency. A small share means that the device waits in its low-power state for almost the whole cycle, so most of its compute capacity is unused and the daily figure is set by the static draw of the board, not by the model. The STM32 H745ZI shows this directly. INT8 export lowers its energy per inference about threefold, from 188.7 mJ to 60.5 mJ, yet its daily energy falls by only 0.3 %. The accelerator rows show the same effect more strongly. The Coral accelerator lowers energy per inference on the Raspberry Pi 4B from 12.6 mJ to 2.4 mJ, which saves about 29 J per day at the 30 s cycle, while it raises the low-power draw by about 1.4 W, which costs about 123 kJ per day. Inference energy and low-power energy are equal at the sampling period $T^* = T_{\text{infer}}(1 + P_{\text{infer}}/P_{\text{low}})$, which is about 3.5 s for the Arduino Nano 33 BLE Sense, 3.8 s for the LOLIN32 and below 0.6 s for every other configuration. Model-side optimization therefore changes daily energy noticeably only when the sampling period approaches T^* . At the 30 s and 5 min periods of an occupancy sensor, the effective levers are the low-power draw of the hardware and a platform that is not oversized for the workload. The

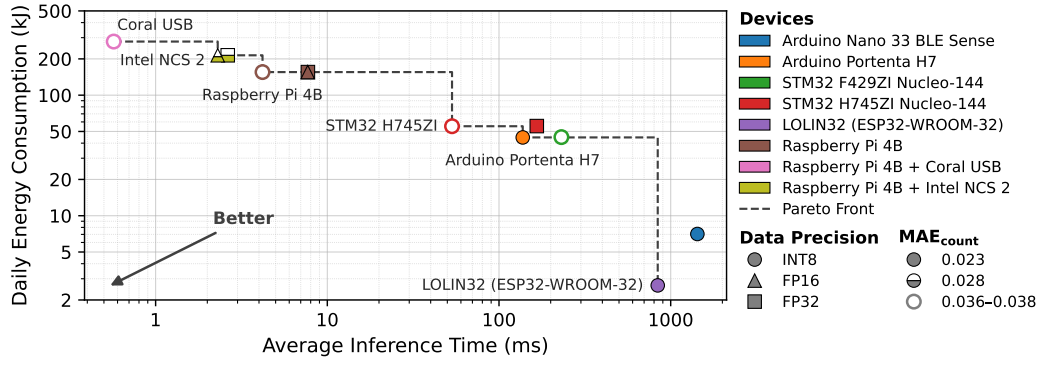


Fig. 3.3. Average inference latency against daily energy at the 30 s sampling period for all configurations of Table 3.6, on logarithmic axes. Marker shape gives the numerical precision, and marker fill gives the counting MAE of Table 3.5. The dashed line is the Pareto front of the two costs, which comprises the Coral accelerator, the Intel NCS2 at FP16, the Raspberry Pi 4B at INT8, the STM32 H745ZI at INT8, the Arduino Portenta H7, and the LOLIN32. The Raspberry Pi 4B FP16 marker coincides with its FP32 marker, and the STM32 F429ZI and the Raspberry Pi 4B at FP32 lie within 0.3 kJ of the front without belonging to it.

active fraction makes the same point in terms of time. Even the slowest board, the Nano 33 BLE Sense, spends 4.8 % of a 30 s period on inference, the LOLIN32 2.8 %, and every other configuration under 1 %.

Figure 3.3 places every configuration of Table 3.6 on the latency and daily-energy plane and draws the front of the two costs. The front runs from the Coral accelerator at 0.57 ms and 278 kJ per day to the LOLIN32 at 840 ms and 2.6 kJ per day, a hundredfold spread in energy across a thousandfold spread in latency, with the Intel NCS2, the Raspberry Pi 4B, the STM32 H745ZI, and the Arduino Portenta H7 in between. It divides into two distinct groups, an accelerator and single-board group above 150 kJ and a microcontroller group below 60 kJ. Because every configuration meets both sampling periods, the position along the front is chosen by the energy budget alone, and integer export moves the H745ZI and the Raspberry Pi 4B only horizontally, to lower latency at the same daily energy.

3.2.3 Conclusions

Deployment feasibility under the most severe form of the memory constraint rests on three coordinated choices, a task formulation that stays meaningful at low fidelity, an operator-simple architecture that keeps activation memory small, and INT8 quantization using the target toolchain. The resulting pipeline fits, compiles, and runs on all five MCU-class targets, holds counting performance covering the annotated occupancy range (Table 3.4), and completes an inference well within both sampling periods. Daily energy follows from the low-power draw, not from inference cost, so the ESP32-based LOLIN32 is the most economical board even though the Cortex-M7 targets infer faster, and Table 3.6 allows an installation to

derive battery lifetime directly once pack capacity and converter efficiency are fixed. At the 30 s cycle the LOLIN32 draws about 0.73 W h per day. The runtime is part of the configuration, since the same integer model reproduces full-precision accuracy on TFLM and loses 0.013 counting MAE on X-CUBE-AI.

Limitations.

- The evaluation uses one sensor type, a small set of mounting configurations, and occupancy levels up to five. At 32×24 pixels, people standing close together merge into one thermal region, and objects near body temperature can produce false detections, so denser scenes and installations near uncontrolled heat sources require site-specific commissioning and additional data.
- Peak activation memory was not recorded, so memory feasibility rests on execution on the 256 kB boards and on the artifact sizes rather than on a measured headroom.
- Power was measured at the development-board level, so the low-power draw of the Portenta and Nucleo boards (515 mW to 640 mW) reflects debuggers and regulators rather than the sleep current of the microcontrollers, and the energy ranking would change for a product design.

3.3 Deployment feasibility and physical consistency in induction motor fault diagnosis

Infrared thermography provides a non-contact source of diagnostic evidence for induction motors [90, 10], and a thermal camera paired with an edge device can classify the operating state continuously without interrupting operation or adding the contact-based sensors that current- and vibration-based diagnostics require [91, 92]. The resulting state class supports inspection prioritization and condition monitoring within predictive-maintenance workflows. This section moves from the memory-constrained microcontrollers of Sec. 3.2 to heterogeneous targets, where the limiting constraint is the toolchain, and adds the physical consistency criterion of Sec. 2.3.1. A model that compiles and runs within budget is not yet trustworthy, because on a controlled test stand the fixtures, cabling, and load device co-occur with the classes they accompany. A sufficiently expressive model can learn the stand rather than the fault. The study therefore treats convertibility as a measured property and pairs cross-runtime benchmarking with attribution inspection against an expectation derived in advance from the physics of each fault.

3.3.1 Method

Test stand, fault classes and expected thermal signatures. Trials are recorded on the laboratory test stand of Fig. 3.4, in which a Workswell WIC 640 camera [93]

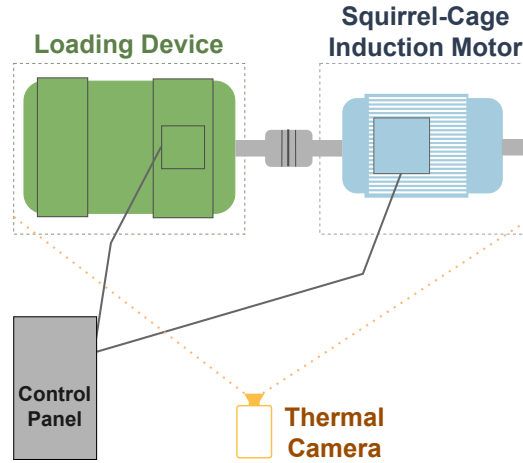


Fig. 3.4. Diagram of the laboratory test bench. A thermal camera observes a 1.5 kW three-phase squirrel-cage induction motor and its coupling to the load apparatus, which is the area that the attribution maps in Fig. 3.6 are expected to highlight.

observes a three-phase squirrel-cage induction motor of 1.5 kW rated power and its coupling to the load apparatus at 640×512 pixels and 4 FPS (Table 2.4). The 42 trials cover three motor states under different current loads and mechanical couplings, and each state has an individual thermal mechanism. Healthy denotes normal operation without a localized hotspot. Angular shaft misalignment, recorded at four ordinal severity settings of the stand, raises friction and therefore power draw, and the dissipated energy heats the housing, the coupling, and the bearings [94]. Broken rotor bars, recorded with one, three, and six severed bars, force the remaining intact bars to carry additional current, which produces asymmetric overheating of the cage that appears as hotspots on the housing [95, 90]. These mechanisms give the expectation used later, namely that a classifier diagnosing misalignment should attend to the coupling and housing, one diagnosing broken bars to the housing, and neither to the load device or the background. Camera position, field of view, and emissivity settings are held constant across trials, so that differences between recordings reflect motor state, load, and coupling rather than instrumentation.

Data split and training. The 5,653 frames are evaluated with five-fold cross-validation, in which each split trains on three folds, uses the fourth for validation and early stopping, and tests on the fifth [13]. Frames within one trial are near-duplicates, because a trial is a 30-second sequence under one fixed configuration, so the meaningful sampling unit is the trial. The published fold sizes, between 832 and 1,695 frames, and healthy-class counts between 186 and 879 per fold show that frames were assigned to folds in groups rather than individually, since frame-level random assignment would give five folds of about 1,131 frames with nearly equal class shares. Each fold reports precision and F1 score weighted by class support (Sec. 2.7.4), and Table 3.7 gives the mean over folds with the fold-to-fold standard deviation. All architectures are trained with the AdamW optimizer [96] at an initial

Tab. 3.7. Classification results of the thirteen candidate architectures under five-fold cross-validation with class weighting [13]. Values are the mean over folds with the fold-to-fold standard deviation.

Architecture	Accuracy $\uparrow$	Precision $\uparrow$	F1 score $\uparrow$
PP-LCNet-0.5	0.924 ± 0.078	0.979 ± 0.045	0.939 ± 0.066
PP-LCNet-0.75	0.920 ± 0.096	0.954 ± 0.072	0.928 ± 0.086
PP-LCNet-1.0	0.880 ± 0.102	0.930 ± 0.053	0.892 ± 0.088
SemNASNet-0.5	0.901 ± 0.091	0.964 ± 0.075	0.920 ± 0.080
SemNASNet-0.75	0.913 ± 0.054	0.979 ± 0.043	0.935 ± 0.045
SemNASNet-1.0	0.878 ± 0.108	0.978 ± 0.043	0.911 ± 0.071
ResNet10t	0.831 ± 0.167	0.952 ± 0.076	0.866 ± 0.136
ResNet18	0.850 ± 0.172	0.960 ± 0.061	0.879 ± 0.136
ResNet34	0.925 ± 0.093	0.957 ± 0.061	0.929 ± 0.086
EfficientNet-B0	0.896 ± 0.141	0.946 ± 0.078	0.907 ± 0.131
EfficientNet-B1	0.932 ± 0.086	0.969 ± 0.046	0.941 ± 0.076
EfficientNet-B2	0.854 ± 0.137	0.899 ± 0.103	0.864 ± 0.129
MixNet-S	0.904 ± 0.088	0.947 ± 0.050	0.916 ± 0.077

learning rate of 3×10^{-5} , a cross-entropy loss, and early stopping with a patience of ten epochs on the validation loss.

Candidate architectures and conversion pipeline. The benchmark compares two lightweight families designed for embedded inference, PP-LCNet [97] and SemNASNet [98], against widely used general-purpose backbones, ResNet [27], EfficientNet [30] and MixNet [99], each at several capacity points, thirteen architectures in all. Deployability is treated as a validation gate across a representative set of edge runtimes [41, 100, 40, 101] on the Raspberry Pi 4B [80] and the Intel NCS2 [82] of Sec. 2.7.3. Models are exported to ONNX and converted to the representation each runtime requires, with FP16 and INT8 variants evaluated wherever the runtime supports them, which is the subgraph-delegation strategy of Sec. 2.2.3 exercised as an experimental variable. The versions are tflite-runtime 2.11.0 and PyArmNN 32.0.0 on the Raspberry Pi CPU, OpenVINO 2022.1 on the NCS2, and ORT 1.14.1 on both, recorded because operator and precision support differ across releases. System measurements are per-frame latencies under a fixed runtime configuration; because a thermal state evolves over minutes, per-frame latency, not a hard deadline, characterizes budget compliance for this workload.

3.3.2 Evaluation

Classification performance. EfficientNet-B1 attains the highest mean accuracy of the thirteen architectures in Table 3.7, 0.932, but its margin over PP-LCNet-0.5 (0.924) and ResNet34 (0.925) is an order of magnitude below the fold-to-fold standard deviation of about 0.08, so the three lead jointly and EfficientNet-B1 serves

Tab. 3.8. Average inference time (s) on edge evaluation hardware, split by execution family (lower is better). Panel A reports TensorFlow Lite and PyArmNN paths; Panel B reports ORT CPU and MYRIAD paths and OpenVINO. Error codes: 1* unresolved custom op (PyFunc); 2* aborted; 3* operator not supported; 4* unsupported HardSwish.

<i>A. TensorFlow Lite and PyArmNN</i>				
Architecture	TFLite FP32	TFLite FP16	TFLite INT8	PyArmNN INT8
PP-LCNet-0.5	0.336 ± 0.006	0.323 ± 0.005	0.228 ± 0.002	0.156 ± 0.003
PP-LCNet-0.75	0.561 ± 0.009	0.560 ± 0.007	0.399 ± 0.003	0.292 ± 0.004
PP-LCNet-1.0	0.764 ± 0.011	0.787 ± 0.009	0.581 ± 0.005	0.513 ± 0.014
SemNASNet-0.5	0.493 ± 0.009	0.449 ± 0.008	0.363 ± 0.002	0.485 ± 0.025
SemNASNet-0.75	0.927 ± 0.050	0.853 ± 0.009	0.745 ± 0.002	1.023 ± 0.016
SemNASNet-1.0	1.083 ± 0.017	1.050 ± 0.009	0.852 ± 0.007	1.102 ± 0.018
ResNet10t	1*	1*	1*	1*
ResNet18	1.725 ± 0.045	2.749 ± 0.021	2*	1.429 ± 0.070
ResNet34	3.204 ± 0.018	5.511 ± 0.027	2*	2.167 ± 0.102
EfficientNet-B0	1.790 ± 0.029	1.722 ± 0.026	1.386 ± 0.006	1.577 ± 0.130
EfficientNet-B1	2.478 ± 0.030	2.487 ± 0.021	1.916 ± 0.004	2.427 ± 0.166
EfficientNet-B2	2.493 ± 0.047	2.635 ± 0.039	2.051 ± 0.008	2.135 ± 0.184
MixNet-S	1.718 ± 0.061	2.109 ± 0.032	1.150 ± 0.024	3*
<i>B. ORT and OpenVINO</i>				
Architecture	ORT CPU FP32	ORT MYRIAD FP16	OpenVINO MYRIAD FP16	
PP-LCNet-0.5	0.156 ± 0.006	4*	4*	
PP-LCNet-0.75	0.253 ± 0.007	4*	4*	
PP-LCNet-1.0	0.353 ± 0.008	4*	4*	
SemNASNet-0.5	0.278 ± 0.113	0.084 ± 0.001	0.083 ± 0.001	
SemNASNet-0.75	0.438 ± 0.026	0.125 ± 0.001	0.125 ± 0.001	
SemNASNet-1.0	0.524 ± 0.025	0.143 ± 0.002	0.142 ± 0.001	
ResNet10t	0.696 ± 0.035	0.127 ± 0.001	0.127 ± 0.001	
ResNet18	0.954 ± 0.030	0.162 ± 0.002	0.161 ± 0.001	
ResNet34	2.216 ± 0.134	0.274 ± 0.001	0.274 ± 0.001	
EfficientNet-B0	0.984 ± 0.046	0.314 ± 0.001	0.313 ± 0.002	
EfficientNet-B1	1.313 ± 0.041	0.456 ± 0.002	0.455 ± 0.002	
EfficientNet-B2	1.377 ± 0.034	0.492 ± 0.002	0.491 ± 0.002	
MixNet-S	0.881 ± 0.021	0.239 ± 0.001	0.238 ± 0.001	

as a representative high-scoring model. Accuracy is only weakly related to capacity. The compact PP-LCNet-0.5 reaches 0.924 with a fraction of the parameters. In contrast, the larger EfficientNet-B2 and ResNet18 fall to 0.854 and 0.850, which corresponds to the observation of the originating study that the task and data do not reward additional capacity [13].

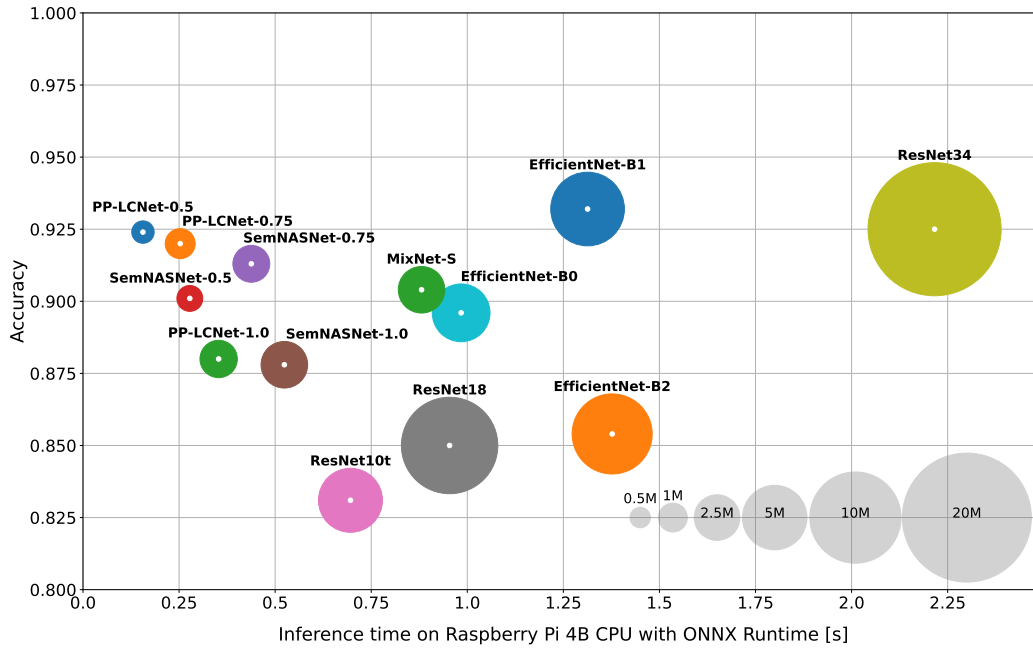


Fig. 3.5. Accuracy, latency and model-size trade-off across the candidate architectures, measured on the SBC baseline of Table 2.6. Marker size encodes the number of parameters, and accuracy is the weighted 5-fold mean of Table 3.7.

Conversion as a feasibility gate. The failures in Table 3.8 show why conversion must precede final model selection. The PP-LCNet family cannot execute on the VPU because HardSwish is unsupported, ResNet10t fails on every TFLite path because of an unresolved custom operator, ResNet18 and ResNet34 abort during INT8 conversion, and MixNet-S meets an unsupported operator in PyArmNN. Four failure types remove seven of the thirteen architectures from at least one execution path, and they affect otherwise competitive candidates. The feasible set, and with it the accuracy–latency trade-off of Fig. 3.5, therefore depends on the runtime. On the Raspberry Pi CPU with ORT, the non-dominated configurations in accuracy and latency are PP-LCNet-0.5 and EfficientNet-B1; on the NCS2, where PP-LCNet cannot run, they are SemNASNet-0.5, SemNASNet-0.75, ResNet34 and EfficientNet-B1. Since the accuracy differences lie within the fold dispersion, the practical conclusion is that PP-LCNet-0.5 and SemNASNet-0.5 lie near the accuracy frontier at a fraction of the latency and number of parameters of the ResNet and EfficientNet families, and that which of the two is available depends on the target.

Runtime selection. For CPU-only operation on the Raspberry Pi 4B, ORT gives the lowest latency for almost every candidate and executes all of them; PyArmNN is slightly faster for ResNet34 and ties for PP-LCNet-0.5 within rounding. For the supported architectures, offloading to the NCS2 through MYRIAD FP16 lowers latency by 2.8 to 8.1× relative to ORT on the CPU, and direct OpenVINO execution and the ORT MYRIAD provider differ negligibly; one interface can therefore serve both targets without a measurable penalty. These measurements describe the 2022–

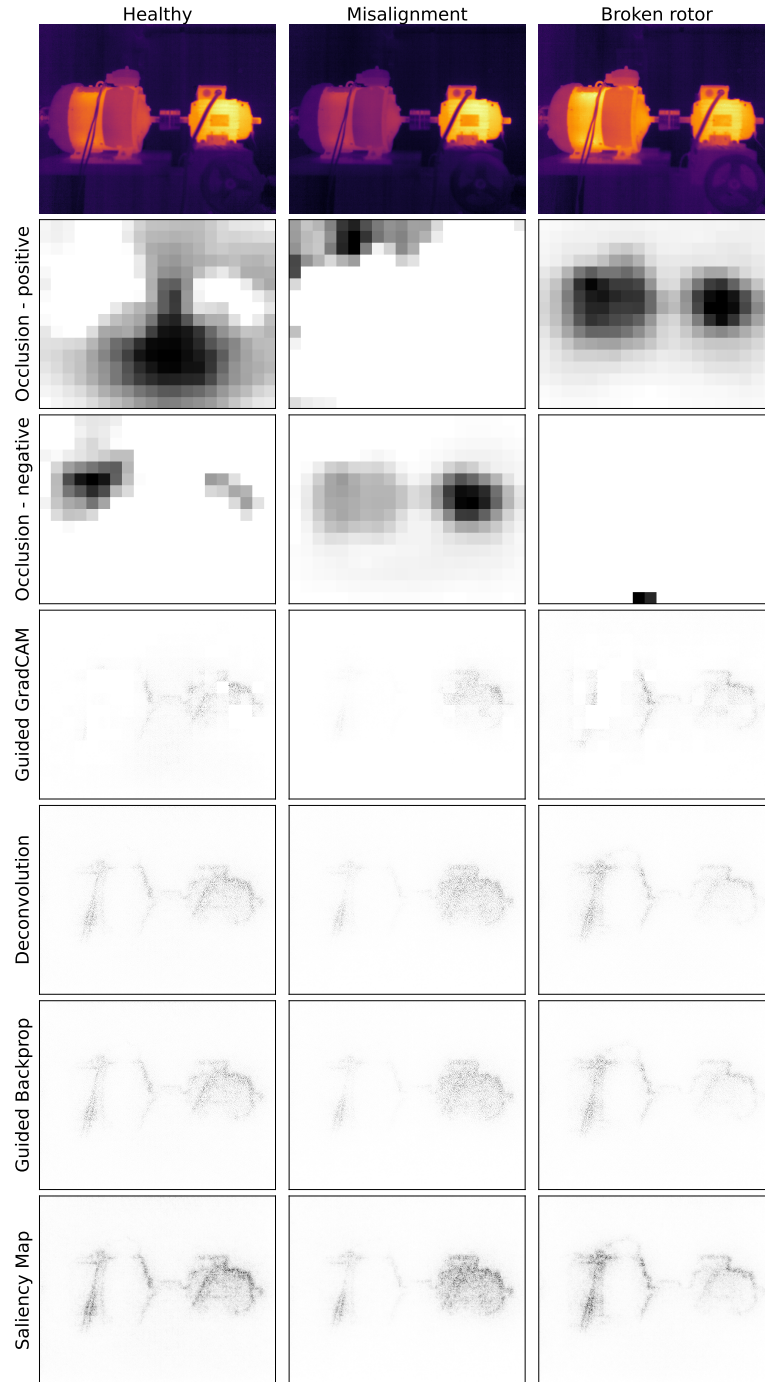


Fig. 3.6. Qualitative attribution inspection of the full-precision motor classifier, using one image per class to compare highlighted regions with the expected fault signatures of Sec. 3.3.1. The comparison concerns which regions are highlighted, not the intensity across panels or methods.

2023 toolchains. Intel has since discontinued the NCS2 and OpenVINO releases after 2022 removed MYRIAD support. Hence, the accelerator results document a feasible set that has already changed, which is itself the reason for measuring convertibility per release rather than assuming it.

Attribution against the expected signatures. EfficientNet-B1 was examined with five attribution methods from the Captum library [102], namely Occlusion [103], Guided Grad-CAM [104], Deconvolution [103, 105], Guided Backpropagation [106] and Saliency [107], on one image per class (Fig. 3.6). Their agreement serves as a consistency test against the expectation of Sec. 3.3.1. Two properties of the methods qualify what agreement can show. Deconvolution and Guided Backpropagation largely reconstruct edges of the input [108] and change little when the weights are randomized [49], so their agreement with the other maps can come from the hot, high-contrast regions of a thermal image; the model-specific evidence rests on Occlusion, which perturbs the input, and on the gradient-based Saliency. For the misalignment class, the maps concentrate on the motor housing and the coupling, as the friction mechanism indicates. For the broken-rotor class, they concentrate on the upper part of the housing, consistent with asymmetric cage heating, but also on the wiring of the load device, which is not part of the predicted signature and functions as a candidate incidental cue. For the healthy class, the attribution spreads over the whole stand, motor, and load device alike, which agrees with the absence of a hotspot being the evidence. The maps were computed for the full-precision model, so the check concerns the trained weights that the converted FP16 and INT8 artifacts of Table 3.8 inherit.

3.3.3 Conclusions

The toolchain determines the feasible set. Backbones that cross-validation would have preferred fail to convert or run under specific runtime and numerical precision combinations (Table 3.8), and the non-dominated configurations differ between the CPU and the accelerator, so conversion is an early screening step, not a late integration step. Among the survivors, the compact PP-LCNet-0.5 and SemNASNet-0.5 match the accuracy of backbones several times larger at a fraction of the latency, and NCS2 offloading brings a further 2.8 to $8.1\times$ where the operator set permits it. The attribution maps of the representative classifier agree with the fault physics for both fault classes and expose the load-device wiring as a cue to be removed by installation design or by masking. This is the qualitative physical-consistency evidence of the thesis; Sec. 6.3 records the region masks, per-region attribution scores, and randomization controls that would turn it into a quantitative acceptance rule.

Limitations.

- The fold assignment file of the evaluation is not preserved. The fold sizes establish group-wise assignment, but trial-level disjointness cannot be confirmed from the surviving records, and the fold-to-fold dispersion is reported descriptively rather than as a test between architectures.

- The benchmark covers one test stand, one camera and one installation geometry, with fault severities recorded as ordinal settings; transfer to other motors, cameras or environments requires re-validation under the same feasibility gates, and thermal sensing itself needs recalibration whenever ambient conditions or surface emissivity change appreciably.
- The attribution check is qualitative, one image per class without region masks, run on the full-precision model rather than on the converted artifact, and the diffuse healthy-class maps do not separate the absence of a hotspot from features of the stand.

3.4 Predictive robustness and its limits in all-sky irradiance forecasting

The two preceding studies evaluate a model at a fixed operating point, with fixed room geometry and occupancy priors in Sec. 3.2 and load and ambient conditions drawn from one trial set in Sec. 3.3. This section analyzes the third criterion of the chapter, predictive robustness, together with the budget clause under a streaming deadline, on intra-hour solar irradiance forecasting from all-sky images. Short-horizon forecasts inform photovoltaic ramp management and storage dispatch [8], and the deadline follows from the one-minute acquisition cadence of the sky imager and the intra-hour decision windows of grid and battery energy storage system (BESS) dispatch [109]. The input stream is highly variable on the time scale of the horizon; the transitions that matter occupy a minority of samples, and saturation near the Sun makes clear sky and optically thin cloud nearly identical in RGB space. Under a fixed budget, robustness obtained from model capacity costs response time and power on every frame. Hence, the section asks how much of it can be obtained instead from input structure, with the preprocessing cost incurred once per sample outside the learned function. The question is whether physics-informed input channels improve prediction within a chosen backbone family, which treats representation as a design choice alongside capacity.

3.4.1 Method

Dataset and model. The pipeline is built and evaluated on the Folsom record [77, 78], with 2014 held out as the test year and 2015 and 2016 used for training and validation, the validation days being held out as whole days (Sec. 2.7.2 and Table 2.4). The model of Fig. 3.7 follows the encoder–head factorization of Eq. 2.2. A ResNet50 encoder [27], initialized with ImageNet-pretrained weights [110], extracts spatial features from the sky-image tensor; the embedding is concatenated with the current irradiance I_t and the readings I_{t-5} , I_{t-10} , and I_{t-15} , and a head of two linear layers, 2052 to 256 with ReLU and 256 to 1, predicts the irradiance at horizon H . The

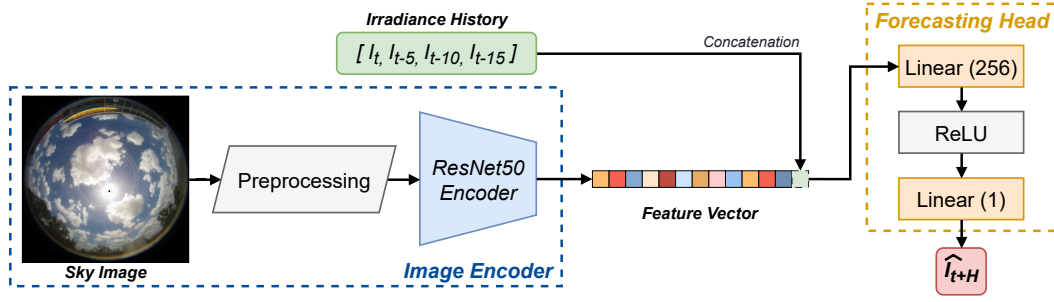


Fig. 3.7. Spatio-temporal irradiance forecasting model. A ResNet50 [27] encoder extracts spatial characteristics from the multi-channel sky-image tensor; these are concatenated with a short history of irradiance measurements and passed to a lightweight regression head predicting irradiance at horizon H .

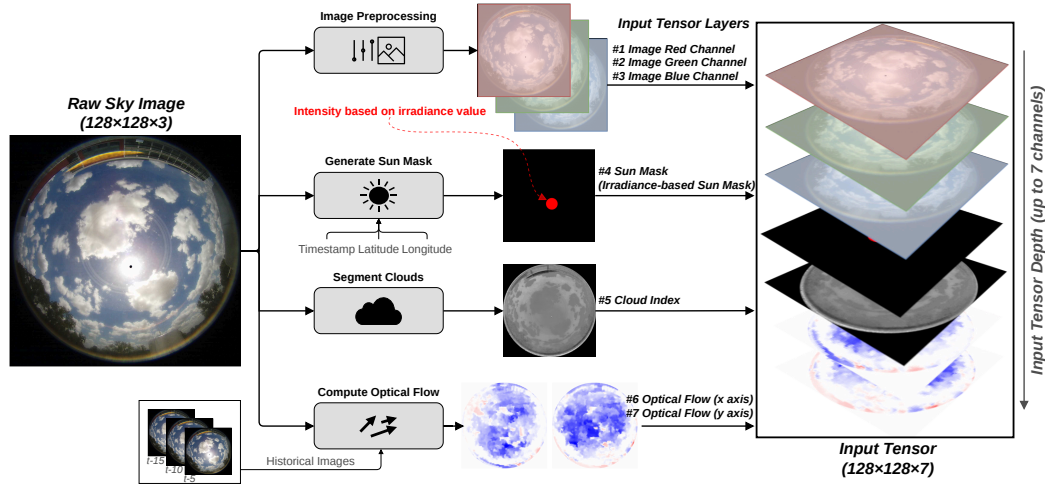


Fig. 3.8. Construction of the multi-channel input tensor. The individual IEP channels are shown alongside the composite result, namely the analytically computed sun mask and its irradiance-valued variant, the dense optical-flow field encoding cloud advection between consecutive frames, and the cloud channel given by normalized red-blue ratio (NRBR) = $(B - R)/(B + R)$. RGB, the irradiance-valued sun mask, the cloud index, and the two flow components form the seven-channel full configuration; the binary sun mask and its irradiance-valued replacement are alternatives, not two simultaneously stacked channels.

encoder carries the parameters and the compute, so the encoder–head interface is where deployment cost is set and where Chs. 4 and 5 later intervene.

Physics-informed input channels. The image enhancement pipeline (IEP) applies the reasoning of Sec. 2.2.2 by computing part of what the network would otherwise have to learn, at a per-sample preprocessing cost visible in T_{pre} . It appends four channels to the recorded RGB frame and leaves the recorded channels unchanged (Fig. 3.8). The sun-position channel encodes where the solar disc lies in the image. It is computed analytically for every frame from the acquisition timestamp and the station coordinates with the solar-position algorithm of pvlb [111, 112] and projected onto the image plane following Papatheofanous et al. [79], so it tracks the diurnal and seasonal motion of the Sun independently of cloud appearance.

The irradiance channel fuses the pyranometer reading into the same tensor. In its first variant, the channel is filled with the current value I_t ; in the second, used in the full configuration, the binary sun mask is replaced by a mask valued with I_t , which allows the encoder to associate the visual state at the solar position with the radiometric quantity it produced instead of meeting the measurement only at the head. The cloud channel exploits a spectral property of the atmosphere. Rayleigh scattering is strongest at short wavelengths, which makes clear sky blue. In contrast, cloud droplets scatter nearly achromatically and appear gray or white, so the red-to-blue ratio (RBR) = R/B [113], the difference $R - B$, and the NRBR = $(B - R)/(B + R)$ [114] give continuous cloud indices, one of which is appended as the cloud index channel without thresholding.

The two motion channels encode cloud advection, the process that governs short-horizon variability, since a forecast at fifteen minutes is largely a question of which clouds will occlude the solar disc by then. Dense optical flow [115, 116] is computed on grayscale frames chronologically from $t - 15$ to t , each pair warm-started from the previous flow, and the final field from $t - 5$ to t supplies the horizontal and vertical displacement components. Each channel therefore converts a quantity the deployment already possesses, a clock and a coordinate, a scattering law, or a pair of consecutive frames, into an explicit input, so the model need not learn the position of the Sun, the spectral characteristic of cloud, or the motion of the sky from data.

Training and loss. Training uses geometric augmentation, namely random translations of up to 10 %, rotations of up to 10°, scaling of up to 10 %, and flips, each applied with probability 0.5 [15], and a batch size of 128, reduced to 96 and 64 at the two largest input resolutions. Large errors during ramp events matter more for the downstream decision than small errors under stable sky, so the training penalty should grow faster than linearly for large residuals while staying linear for small ones. This is the behavior of the mean adaptive BerHu (reverse Huber) penalty [117]. For a mini-batch $\mathcal{B}$ with residuals $e_t = \hat{y}_t - y_t$,

$$\mathcal{L}_{\text{BerHu}} = \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \rho_T(e_t), \quad \rho_T(e) = \begin{cases} |e|, & |e| \leq T, \\ \frac{e^2 + T^2}{2T}, & |e| > T, \end{cases} \quad (3.5)$$

where the threshold is recomputed for every batch from its largest residual,

$$T = 0.2 \max_{t \in \mathcal{B}} |e_t|. \quad (3.6)$$

The two branches meet at $|e| = T$ with equal value and equal slope, so ρ_T is continuously differentiable. Below T , the gradient has constant magnitude, as for MAE, and above T it grows in proportion to the error, as for MSE; the Huber and

SmoothL1 penalties do the opposite. Because T follows the largest residual in each batch, the switch point adapts to the error range of that batch instead of being fixed in advance. The loss is selected before any input enhancement is applied, and the evaluation holds it fixed for every subsequent experiment.

3.4.2 Evaluation

Loss function and input representation. Table 3.9 isolates the penalty first and the input representation second, at the 15-minute horizon and 128×128 input. With the pretrained encoder, BerHu leads the four conventional penalties by 1.41 forecast skill points and 1.38 W/m^2 MAE over the strongest alternative in each metric; with random initialization, it has the lowest MAE while MSE has the higher forecast skill, so the advantage of BerHu is realized together with pretraining. Pretraining itself accounts for 2.57 W/m^2 and 4.53 skill points at fixed loss, and it is therefore treated as a separate factor.

Table 3.10 presents the sweep as one sequence of additions. Along it, the 6.15 W/m^2 MAE improvement from the randomly initialized model to the full configuration decomposes into 2.57 from pretraining, 1.68 from augmentation and 1.91 from the four channels, and the 10.34-point forecast skill gain into 4.53, 1.98 and 3.83. After pretraining is fixed, the channels contribute 3.83 of the 5.81 skill points, about two-thirds. These are increments along one order of addition, and the channels are not independently additive. The constant irradiance channel gives 17.77 skill against 17.88 for augmentation on its own, whereas its combination with the sun mask, which localizes the reading at the solar position, improves the score. Dense Inverse Search gains 1.44 points while Farnebäck loses 1.21. NRBR is the weakest isolated cloud channel at 16.19, yet inside the complete pipeline the three cloud variants lie within 0.47 points of each other, where the normalization by $B + R$ plausibly removes exposure variation that matters once the sun mask is present. Selecting channels only in isolation would therefore reject a useful combination. The full configuration with DIS flow and NRBR is carried forward; it has the lowest RMSE, shares the lowest nRMSE with the $R - B$ variant, and has the highest skill of the three cloud variants, but the single-run design cannot resolve differences of that size, so the choice rests on the normalization argument above rather than on the ranking.

Representation versus capacity. Nineteen backbone architectures [27, 30, 118, 119, 120] compare input structure with capacity (Fig. 3.9). The unenhanced configurations are randomly initialized, whereas the enhanced configurations receive the full recipe of pretraining, augmentation, and the four channels. Hence, the sweep measures the joint effect of the recipe on each backbone. The recipe improves MAE, RMSE, and forecast skill for all nineteen backbones and compresses the differences between them. The MAE spread narrows from 34.1 W/m^2 to 3.0 W/m^2 and

Tab. 3.9. Loss-function and input-representation ablation for the ResNet50 backbone at a 15-minute horizon. The upper blocks isolate the penalty before any input enhancement, first with a randomly initialized encoder and then with ImageNet-pretrained weights, and the pretrained BerHu row is the unenhanced baseline for the lower block, in which BerHu is held fixed, and augmentations are held fixed from the second row onward. FS is measured against the Smart Persistence reference of Eq. 2.10, which is therefore zero for it by construction. Best value per column in bold. DIS denotes Dense Inverse Search optical flow [115]; Farnebäck denotes [116]. Cloud variants are R/B , $R - B$, and $NRBR = (B - R)/(B + R)$. The irradiance flag alone denotes the constant irradiance channel; together with the sun-mask flag, it denotes the single irradiance-valued mask of the full configuration, not two extra tensor channels.

Loss	Aug.	Sun mask	Irrad.	Flow	Cloud	MAE (W/m ²)↓	MAPE (%)↓	RMSE (W/m ²)↓	nRMSE (%)↓	FS (%)↑
<i>Reference</i>										
Smart Persistence	–	–	–	–	–	36.914	17.14	88.176	21.36	0.00
<i>Loss function, no input enhancement, random initialization</i>										
MSE						40.809	24.05	76.021	18.42	13.79
MAE						38.813	18.77	83.457	20.22	5.35
SmoothL1						37.614	19.54	79.576	19.28	9.75
Huber						36.755	18.22	80.925	19.61	8.22
BerHu						35.347	17.87	78.153	18.94	11.37
<i>Loss function, no input enhancement, ImageNet-pretrained encoder</i>										
MSE						39.569	24.11	75.396	18.27	14.49
MAE						34.542	18.01	76.544	18.55	13.19
SmoothL1						34.158	18.43	77.388	18.75	12.24
Huber						34.711	18.66	78.107	18.93	11.42
BerHu						32.780	17.95	74.156	17.97	15.90
<i>Input representation, BerHu penalty throughout</i>										
	✓					31.103	15.61	72.410	17.55	17.88
<i>Geometry and radiometry</i>										
	✓	✓				30.526	14.49	72.055	17.46	18.28
	✓		✓			30.623	15.88	72.509	17.57	17.77
	✓	✓	✓			29.633	15.49	71.539	17.33	18.87
<i>Motion</i>										
	✓			DIS		29.905	17.98	71.142	17.24	19.32
	✓			Farnebäck		30.888	16.90	73.474	17.80	16.67
BerHu	✓	✓	✓	DIS		29.409	15.41	69.961	16.95	20.66
	✓	✓	✓	Farnebäck		29.857	14.42	72.351	17.53	17.95
<i>Cloud segmentation</i>										
	✓				R/B	30.242	17.00	72.057	17.46	18.28
	✓				$R - B$	30.677	17.32	72.817	17.64	17.42
	✓				NRBR	30.776	16.95	73.903	17.91	16.19
	✓	✓	✓		R/B	30.000	14.78	71.904	17.42	18.45
	✓	✓	✓		$R - B$	30.074	14.57	72.263	17.51	18.05
	✓	✓	✓		NRBR	29.996	15.75	71.745	17.38	18.63
<i>Full pipeline</i>										
	✓	✓	✓	DIS	R/B	29.039	15.09	69.450	16.83	21.24
	✓	✓	✓	DIS	$R - B$	29.052	15.29	69.052	16.73	21.69
	✓	✓	✓	DIS	NRBR	29.198	15.01	69.037	16.73	21.71

the forecast-skill spread from 20.8 to 5.9 points, every enhanced backbone has a lower MAE than every unenhanced one, and the weaker starting points gain most, with a Spearman rank correlation of 0.98 between unenhanced MAE and its reduction. Most of the unenhanced spread comes from MobileNetV4-M, EfficientNet-B0 and EfficientNet-B2, whose randomly initialized runs have negative forecast skill. EfficientNet-B0, with 4.34 million parameters and 0.27 billion FLOPs, moves from

Tab. 3.10. Stepwise decomposition of the ResNet50 improvement at the 15-minute horizon, taken along the rows of Table 3.9 with BerHu held fixed. Each increment is the change from the preceding row; the increments depend on the order of addition.

Configuration	MAE (W/m ²)	Δ MAE (W/m ²)	FS (%)	Δ FS (points)
Random initialization	35.347	–	11.37	–
+ ImageNet pretraining	32.780	–2.57	15.90	4.53
+ Augmentation	31.103	–1.68	17.88	1.98
+ Four IEP channels	29.198	–1.91	21.71	3.83
Total		–6.15		10.34

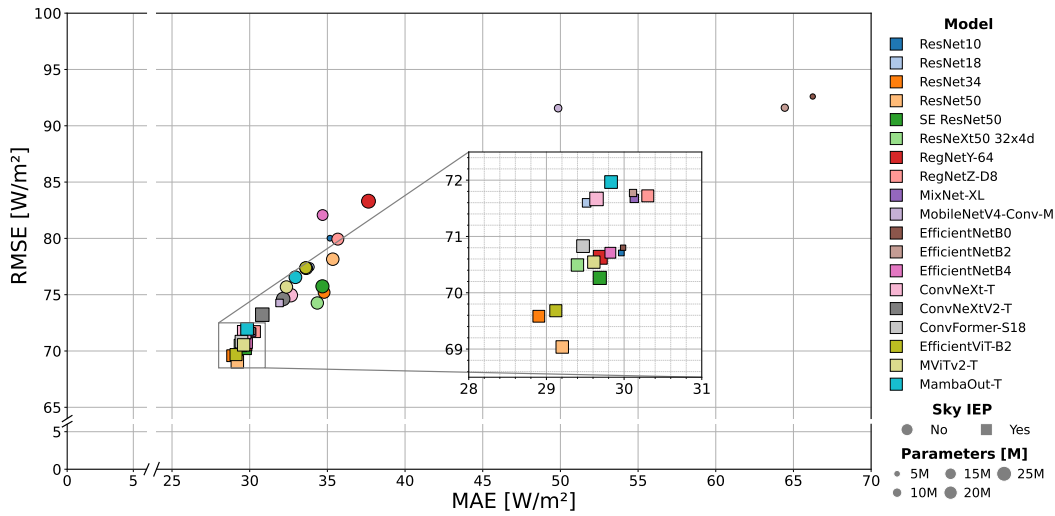


Fig. 3.9. Backbone comparison with and without the IEP, plotting MAE against RMSE for the 19 evaluated architectures. Marker size encodes the number of parameters in millions, and marker shape distinguishes enhanced (squares) from unenhanced (circles) configurations. Enhanced configurations are ImageNet-pretrained, augmented, and given the four physics-informed channels; unenhanced configurations are randomly initialized.

66.242 W/m² MAE and –5.02 % forecast skill to 29.987 W/m² and 19.71 %. The channel gain in the ResNet50 ablation, 3.83 skill points, is of the same order as the forecast-skill spread between the enhanced architectures, 5.9 points, which supports input structure as a design variable alongside backbone choice. Two exact comparisons locate ResNet50 in the plot. Enhanced ResNet34 has a lower MAE (28.902 W/m²) but lower skill (21.09 versus 21.71 %), and EfficientViT-B2 [121] reaches similar MAE (29.121 W/m²) and 20.98 % skill at 1.03 billion FLOPs compared with 2.78 billion for ResNet50, which makes it a candidate for a tighter budget. ResNet50 is nevertheless carried into Chs. 4 and 5, because the reduced-precision execution paths of Sec. 4.4 handle plain convolutional networks better than attention mechanisms and wide-range activations, so the backbone is chosen together with the execution path rather than on prediction error alone.

Tab. 3.11. Comparison against published sky-image forecasting methods on the Folsom test split at a 15-minute horizon. Each comparator was reimplemented following the procedure of its own publication, with input and output formats adapted to those of Sec. 3.4.1, so the values are not the ones reported in the source papers. Linear and Lasso regressors, trained on irradiance history alone, reach a forecast skill of 0.07 % and are omitted. Best value per column in bold, second best underlined.

Method	MAE (W/m ²)↓	MAPE (%)↓	RMSE (W/m ²)↓	nRMSE (%)↓	FS (%)↑
Smart Persistence	36.914	17.14	88.176	21.36	0.00
MLP regressor [122]	38.340	23.56	80.305	19.46	8.93
Wen et al. [123]	41.328	23.54	77.861	18.87	11.70
Zang et al. [124]	41.130	22.46	79.573	19.28	9.76
Mercier et al. [125]	52.039	43.97	84.523	20.48	4.14
Jonathan et al. [126]	54.251	38.86	84.110	20.38	4.61
Ansong et al. [127]	43.069	25.04	76.113	18.44	13.68
Hendrikx et al. [128]	37.915	24.46	74.066	17.95	16.00
Papatheofanous et al. [79]	33.002	<u>15.55</u>	73.982	17.93	16.10
Venitourakis et al. [129]	<u>31.323</u>	15.74	<u>72.908</u>	<u>17.67</u>	<u>17.32</u>
ResNet50 + IEP (this work)	29.198	15.01	69.037	16.73	21.71

Comparison with published methods. On the same Folsom test split and 15-minute horizon, the carried-forward configuration is compared with sky-image forecasters reimplemented from their publications and adapted to this input and output format (Table 3.11). The enhanced ResNet50 leads on every metric, ahead of the XceptionLSTM model of Venitourakis et al. [129] and the edge-oriented model of Papatheofanous et al. [79], by 2.1 W/m² MAE and 4.4 skill points over the strongest comparator. Regressors trained on irradiance history alone mark the value of imagery, since the MLP reaches 8.93 % forecast skill and the linear models none.

Forecast horizons. Table 3.12 evaluates the configuration at horizons of 5, 10, 15, and 30 minutes against Smart Persistence, the history-only MLP regressor [122] and the identical backbone without enhancement. The enhanced pipeline improves on all three comparators at all four horizons and on all metrics, at 128 × 128 input resolution. Its forecast skill relative to Smart Persistence rises with the horizon, from 14.45 % at 5 minutes to 21.71 % at 15 and 22.52 % at 30, as expected, since persistence degrades fastest where the atmospheric state changes most and leaves the most headroom for a learned model. The margin over the unenhanced backbone is 3.51, 7.25, 10.34 and 6.66 skill points, so the advantage is not specific to the horizon at which the recipe was tuned. The history-only regressor degrades fastest, from 9.52 % to 4.04 %, and is worse than persistence on MAE at every horizon while retaining positive skill, because skill is RMSE-based and weights the large errors that imagery helps to avoid. The larger 384 × 384 input leads the smaller one by

Tab. 3.12. Forecast horizon sweep on the Folsom test split. IEP denotes the image enhancement pipeline of Sec. 3.4.1, which constructs the all-sky input channels, in the configuration carried forward, including pretrained weights and the BerHu penalty. MAE and RMSE in W/m^2 ; MAPE, nRMSE (Eq. 2.8), and FS in %. FS is computed against the Smart Persistence row of the same horizon by Eq. 2.9, so that row has no skill value. Lower is better for all metrics except FS. Best value per column within each horizon block in bold.

Horizon	Configuration	IEP	MAE	MAPE	RMSE	nRMSE	FS
5 min	Smart Persistence		25.398	11.03	72.013	17.45	0.00
	MLP regressor		28.663	11.48	65.161	15.79	9.52
	ResNet50 _{128×128}		24.766	11.39	64.137	15.54	10.94
	ResNet50 _{128×128}	✓	22.411	10.20	61.604	14.93	14.45
	ResNet50 _{384×384}	✓	22.358	11.10	60.331	14.62	16.22
10 min	Smart Persistence		32.669	14.78	83.545	20.24	0.00
	MLP regressor		36.615	18.48	75.056	18.19	10.16
	ResNet50 _{128×128}		30.460	15.24	71.539	17.33	14.37
	ResNet50 _{128×128}	✓	26.392	14.06	65.486	15.87	21.62
	ResNet50 _{384×384}	✓	26.405	12.86	64.531	15.64	22.76
15 min	Smart Persistence		36.914	17.14	88.176	21.36	0.00
	MLP regressor		38.340	23.56	80.305	19.46	8.93
	ResNet50 _{128×128}		35.347	17.87	78.153	18.94	11.37
	ResNet50 _{128×128}	✓	29.198	15.01	69.037	16.73	21.71
	ResNet50 _{384×384}	✓	29.338	15.29	68.186	16.52	22.67
30 min	Smart Persistence		46.078	21.82	98.630	23.90	0.00
	MLP regressor		52.025	44.82	94.648	22.93	4.04
	ResNet50 _{128×128}		42.062	28.17	82.991	20.11	15.86
	ResNet50 _{128×128}	✓	35.245	19.47	76.424	18.52	22.52
	ResNet50 _{384×384}	✓	36.591	21.34	79.622	19.29	19.27

Tab. 3.13. Effect of input resolution on 15-minute-horizon forecasting ability for the enhanced ResNet50 configuration. Bold marks the best value in each column. Training batch size was reduced at the two largest resolutions (to 96 and 64 respectively) to remain within training memory; the corresponding inference latency for the bracketing resolutions is reported in Table 3.14.

Image size	MAE (W/m^2)↓	MAPE (%)↓	RMSE (W/m^2)↓	nRMSE (%)↓	FS (%)↑
96 × 96	29.250	14.07	69.652	16.88	21.01
128 × 128	29.198	15.01	69.037	16.73	21.71
192 × 192	29.235	15.24	69.436	16.82	21.25
256 × 256	29.457	14.22	70.778	17.15	19.73
384 × 384	29.338	15.29	68.186	16.52	22.67

1.77, 1.14 and 0.96 skill points at 5, 10, and 15 minutes and loses 3.25 points to it at 30.

Input resolution and time budget. Resolution was varied from 96×96 to 384×384 at the 15-minute horizon (Table 3.13). Forecast quality is nearly flat, with MAE varying by under 0.3 W/m^2 across the range and the largest resolution attaining the

Tab. 3.14. Stage-wise latency in milliseconds on Raspberry Pi 4B and Raspberry Pi 5, decomposed into model inference, preprocessing, and each enhancement channel, at the two bracketing resolutions of Table 3.13. Values are means over the evaluation set with standard deviations; lower is better. The irradiance channel, below 2.4 ms at every setting, is omitted.

Device	Image size	Inference (ms)	Preprocessing (ms)	Sun mask (ms)	Cloud channel (ms)	Optical flow (ms)
RPi 4B	128 × 128	127.28 ± 4.26	22.16 ± 1.04	15.04 ± 0.18	1.44 ± 0.07	10.64 ± 0.62
RPi 4B	384 × 384	1058.64 ± 18.54	78.82 ± 7.83	21.59 ± 0.92	20.00 ± 0.89	80.87 ± 4.13
RPi 5	128 × 128	50.43 ± 4.49	7.89 ± 1.07	6.13 ± 1.08	0.69 ± 0.11	5.35 ± 0.80
RPi 5	384 × 384	455.36 ± 5.88	47.50 ± 3.42	10.83 ± 0.71	11.18 ± 0.49	54.63 ± 2.69

best RMSE marginally, whereas Table 3.14, which resolves the end-to-end latency of Eq. 2.1 into preprocessing, the enhancement channels, and inference, shows inference latency rising by 8 to 9× over the same interval. The deadline is never reached on the measured hosts. At 128 × 128, the Raspberry Pi 5 completes all timed stages in about 70 ms and the Raspberry Pi 4B in about 177 ms, and even the slowest pipeline, 384 × 384 on the Raspberry Pi 4B, finishes in about 1.26 s against a 60 s acquisition period. At 128 × 128, the enhancement channels add 18 % to 21 % to the image-to-irradiance processing time, the sun mask and the optical flow being the two costly stages; at 384 × 384 the optical flow and the inference dominate. The resolution sweep therefore changes how long the host is occupied, not whether it meets the deadline. The active fraction of the period is 0.12 % for the carried-forward configuration on the Raspberry Pi 5 and 2.1 % for the slowest one on the Raspberry Pi 4B, and this slack is the compute that Ch. 4 spends on adapting the model between frames. The lower resolutions are the suitable choice given the small differences in MAE.

Behavior across sky regimes. The difficulty of the task is regime-dependent, and Fig. 3.10 makes the dependence visible for four configurations at the 15-minute horizon. Under clear conditions, the irradiance trajectory is smooth and dominated by the deterministic diurnal component, and every configuration tracks it closely; under overcast conditions, the trajectory is depressed but comparatively stable. The discriminating case is the partly cloudy day, where advecting cloud produces rapid ramps as the solar disc is occluded and revealed, and where the enhanced ResNet50 follows transitions that the unenhanced backbone, the history-only regressor, and the XceptionLSTM forecaster of Venitourakis et al. [129] lag or miss. These transitions determine the annual averages reported in the tables.

Transfer to an unseen site. Whether a model trained at one site is usable at another without further data is itself a deployment question, and Table 3.15 answers it for the carried-forward configuration. The target is the eight-day Poznań collection of Table 2.4, recorded in late October with a different camera and prepared with the same preprocessing and quality control as the Folsom data. Applied zero-shot at the

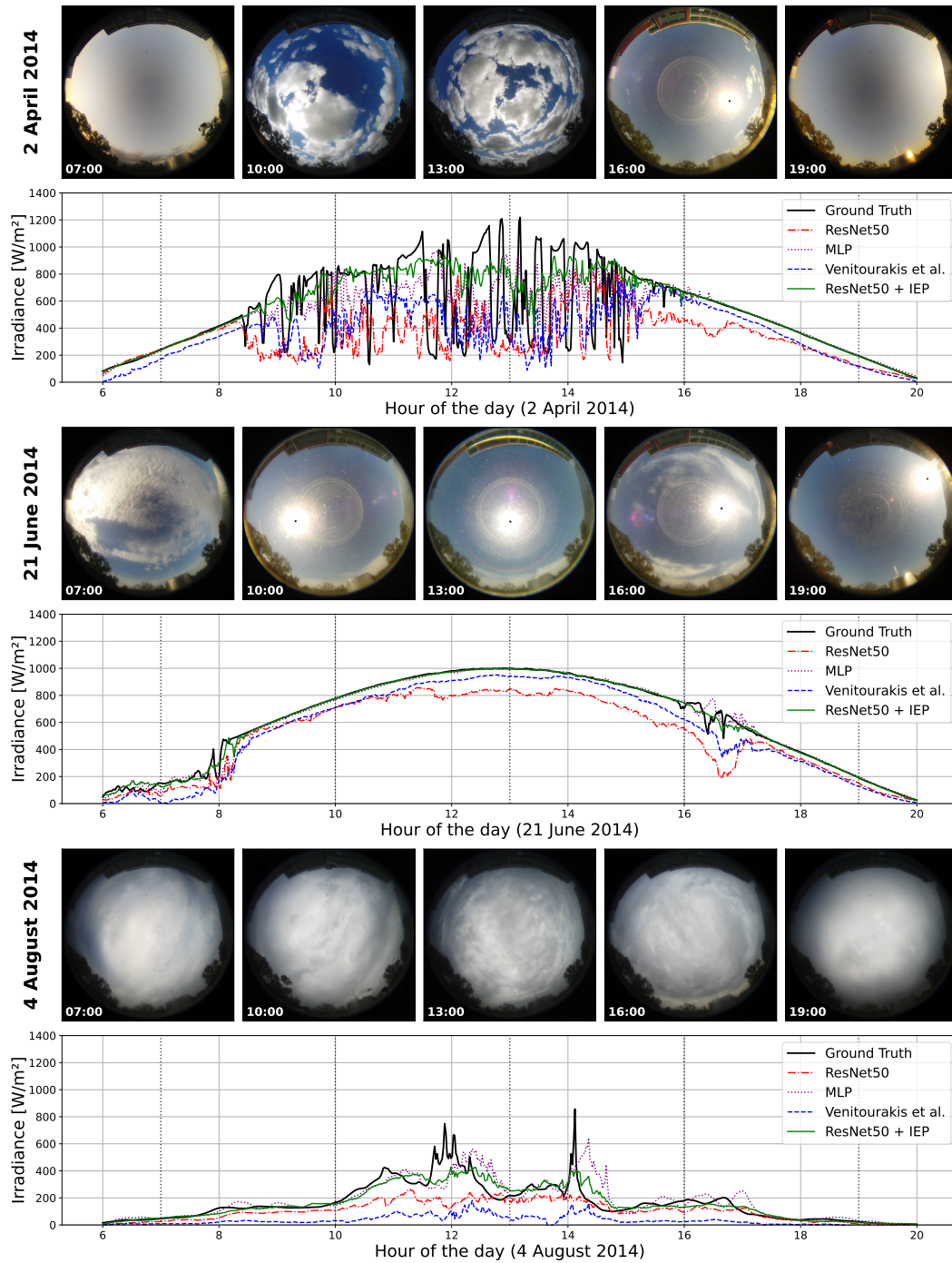


Fig. 3.10. Forecast trajectories over three days with distinct sky conditions, namely partly cloudy (top), clear (middle), and overcast (bottom). Each panel plots measured irradiance against the forecasts of four compared configurations at a 15-minute horizon, namely the unenhanced ResNet50, the history-only MLP regressor, the XceptionLSTM forecaster of Venitourakis et al. [129], and the enhanced ResNet50 with IEP.

Tab. 3.15. Cross-site transfer of the configuration carried forward, at a 15-minute horizon. All Poznań rows are evaluated on the same seven-day test period drawn from an eight-day collection; the few-shot model is adjusted on the excluded day. MAE and RMSE are in W/m^2 , lower is better. FS is computed against Smart Persistence at the evaluation site. Hence, a negative value denotes higher RMSE than the local statistical reference, and the Folsom block is a reference against a different persistence baseline rather than a directly comparable score.

Site	Adaptation	Test days	MAE ↓	RMSE ↓	FS (%)↑
Poznań	Smart Persistence (local)	7	27.405	56.657	0.00
	Zero-shot transfer	7	57.083	76.098	−34.31
	Few-shot, 1 day	7	32.632	53.089	6.30
Folsom	Smart Persistence (local)	—	36.914	88.176	0.00
	Training site (reference)	—	29.198	69.037	21.71

15-minute horizon, the pipeline is substantially worse than local Smart Persistence on both error measures, with a forecast skill of −34.31 %. Fine-tuning on a single day and testing on the remaining seven recovers the model to 6.30 %, far below the 21.71 % attained at the training site. The recovery is narrower than the forecast skill figure suggests, since the adapted model beats local persistence on RMSE (53.089 versus 56.657) while remaining 5.2 W/m^2 worse on MAE. Within the limits of a one-week test at one site, the conclusion is consequential for the remainder of the thesis. A pipeline shown to be feasible, accurate, and robust across horizons and regimes at one installation can be worse than the local statistical reference at another, and validation at the source site does not quantify that risk. The longer campaign of Ch. 4 finds a milder loss at the same site for a differently trained forecaster of the same backbone (Sec. 4.3.1 and Table 4.3), so the size of the loss varies with configuration and season. What the present test establishes is that relocation can fail severely.

Operational value under storage firming. To translate forecast error into an operating quantity, a time-domain simulation of BESS firming over the full Folsom test split compares the proposed forecaster with Smart Persistence (Fig. 3.11), following the setup of the originating study [15]. The storage is dispatched under a 15-minute firming regime to compensate for the difference between the submitted forecast and the realized generation of a nominal 1 MW plant with 92 % round-trip efficiency, and energies are expressed in per-unit hours relative to the nominal capacity, so that one per-unit hour corresponds to 1 MWh for this plant. Two quantities summarize the year. Battery throughput, the charge and discharge energy processed by the storage and a common proxy for cycling wear, falls by 20.8 %, from 2,250.4 to 1,783.3 per-unit hours. Pre-dispatch curtailed energy, the generation schedules rejected for ramp-rate violations, falls by 98.7 %, from 52.33 to 0.67 per-unit hours. The mechanism is visible in the battery trajectories of Fig. 3.11. Persistence reacts to a ramp after it has begun, so the storage first discharges as generation drops and

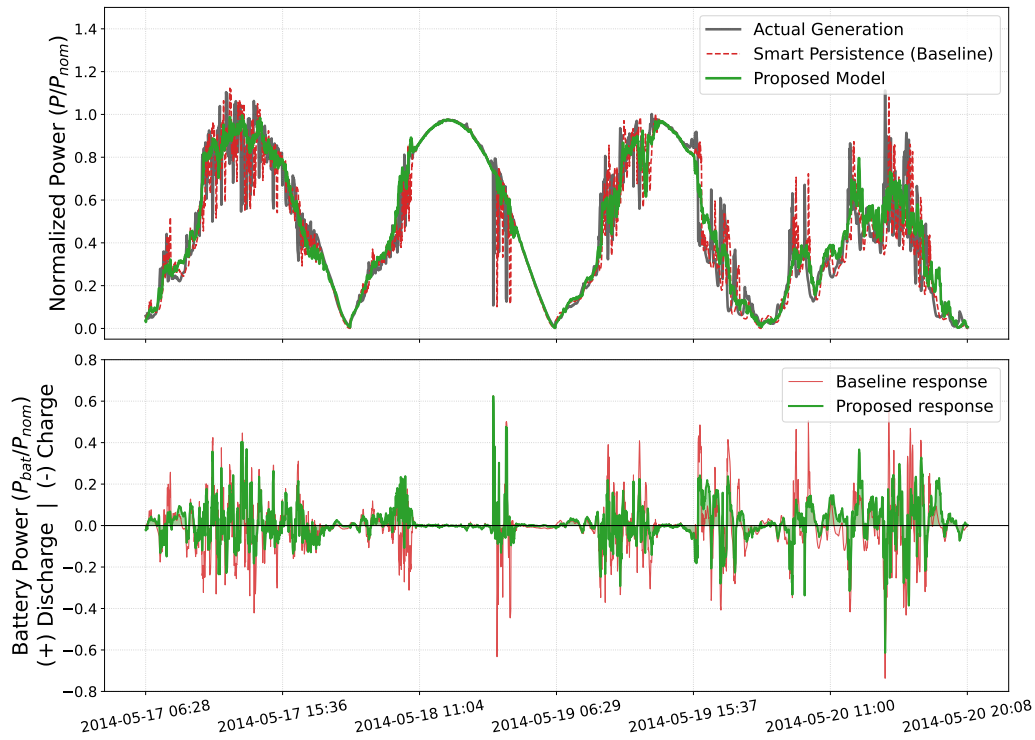


Fig. 3.11. Time-domain simulation of storage firming on the Folsom test split. Top, normalized generation trajectories of the plant together with the Smart Persistence and image-based forecasts; bottom, the battery power required to firm each forecast. The reactive persistence forecast induces repeated charge–discharge cycles at every ramp, whereas the image-based forecast yields a smoother response.

then charges rapidly as the lagged forecast catches up, the double-penalty pattern of reactive dispatch [130], whereas the image-based forecaster anticipates the ramp and the storage responds once. The comparison evaluates the complete forecaster against persistence, so it measures the operational value of the pipeline as a whole rather than of any single channel.

3.4.3 Conclusions

Physics-informed input channels raise forecast skill within a fixed backbone family, and the controlled ResNet50 sequence attributes 3.83 of the 5.81 skill points gained after pretraining to the channels, at a preprocessing cost that stays inside the acquisition interval on both single-board hosts (Tables 3.9 and 3.14). The gain holds across horizons from 5 to 30 minutes and across input resolutions, exceeds the reimplemented published forecasters, and translates into fewer storage cycles and almost no ramp-rate curtailment in the firming simulation. The same configuration fails at an unseen site, where its zero-shot forecast skill is -34.31% and one day of local data recovers only a fraction of the source-site skill. Predictive robustness established on the validation distribution therefore does not extend to a relocated

deployment, and a fixed model cannot recognize when its deployment has left that distribution, a limit whose consequences Sec. 3.5 develops.

Limitations.

- The comparisons rest on single training runs, and the reimplemented comparators were run at their published hyperparameters without tuning to this input format, so differences of a few tenths of a point, such as those between the three cloud channels in the full pipeline, lie within unmeasured run-to-run variability.
- The cross-site test covers one week in late October at one site with a different camera, so it indicates that transfer can fail severely without characterizing how the loss varies with season or with the distance between climates and instruments.
- The storage simulation uses one dispatch rule and one set of plant parameters. It reports throughput as a cycling proxy rather than as a lifetime or cost estimate, and the hardware evaluation covers SBC-class hosts rather than the microcontroller and accelerator targets of Secs. 3.2 and 3.3.

3.5 Summary and the limits of static deployment

The three case studies expose different deployment constraints, memory in Sec. 3.2, toolchain compatibility in Sec. 3.3, and the time budget in Sec. 3.4, and each yields a finding that carries over to the later systems. Within the evaluated settings, five lessons are supported.

1. Convertibility must be measured rather than assumed, since the feasible set and the non-dominated configurations changed with the runtime (Sec. 3.3).
2. Under duty-cycled sampling, daily energy is set by the low-power draw, not by inference cost. The ranking of platforms by energy per inference reverses over a day, model-side optimization reduces daily energy only when the sampling period approaches a few seconds, and a small inference share signals an idle platform rather than an efficient one.
3. Input structure is a design variable alongside backbone choice, initialization, augmentation, and loss. In the ResNet50 ablation, its gain is of the same order as the spread between enhanced architectures, with the preprocessing cost incurred once per sample outside the learned function.
4. Domain knowledge serves twice, as explicit input features for the forecaster and as an independent expectation against which the motor classifier's attribution is inspected; the two roles are assessed by different kinds of evidence, quantitative ablation and qualitative consistency.
5. Validation at one site quantifies neither the risk of relocation nor the loss that follows it.

Evidence for RQ1 and RQ2. The occupancy pipeline improves counting performance with fewer parameters than its reference and runs on kilobyte-scale targets; the motor study shows which architecture and runtime combinations execute and supplies qualitative physical-consistency evidence; and the forecasting ablation quantifies the predictive value of task-specific input structure. These outcomes support the task-specific representation and hardware-aware execution parts of the hypothesis for the tested configurations, and the duty-cycle measurements add that deployment cost depends on the platform and the operating pattern together, not on model size or inference latency alone. The chapter thereby fixes the design dimensions that Chs. 4 and 5 inherit, supported operators and precision, the host–accelerator partition, the encoder–head interface, and the input representation.

The limit of the static configuration. The primary finding of the chapter is the transfer result of Table 3.15. A forecaster validated over horizons, regimes, and resolutions at its training site was worse than the local statistical reference at an unseen site, and one day of local data recovered only a fraction of its source-site skill (Table 3.15). Every configuration in this chapter is fixed before installation and evaluated on pre-installation data, so none can notice that its deployment has left the validated distribution or use the idle time that Secs. 3.2 and 3.4 leave to correct itself. Two responses define the remaining chapters. Chapter 4 lets the deployed forecaster update itself from the delayed supervision the workload provides, inside the compute slack of Table 3.14, and asks which inputs and updates justify that cost. Chapter 5 asks what several such sites can share when one site’s data cannot close the gap alone, exchanging compact regime summaries across the encoder–head interface fixed here.

Reliable adaptation on the deployed device

The predictive performance of a deployed model deteriorates as operating conditions change, and Ch. 3 closed with a forecaster that fell below local persistence at an unseen site. This chapter develops error-triggered adaptation and its limits (C3) on visual irradiance forecasting and on image-path fault screening, and it measures what adaptation costs on the devices of Table 2.6. Three difficulties shape the adaptation protocol of Sec. 2.6.1. Learning shares resources with real-time inference [86], a corrupted observation may or may not be usable as evidence, and supervision arrives late, since the irradiance a forecast refers to is measured only after the horizon has elapsed. Section 4.2 evaluates an error-triggered update policy over recorded streams of 213 to 362 scored days at three sites and finds that a reduced-step partial-encoder action, which updates the last two encoder stages at one tenth of the head learning rate, improves on the static model at every site, whereas the existing full-encoder action deteriorates within months at the source site. Section 4.3 places an image-fault monitor in front of that policy and finds that detecting a fault does not justify withholding the frame from learning, since the tested persistent faults remained useful evidence while their labels stayed valid. Section 4.4 measures, on separate classification workloads, how much an inference accelerator lowers the cost of the head updates that make up the majority of the policy’s updates. The first two studies address RQ3. The third contributes to RQ2 and to the hardware-aware execution contribution C4. The three studies build on the author’s publications [17, 16, 18].

4.1 Adaptation protocol and evaluation setup

4.1.1 The adaptation loop

The deployed device operates a streaming loop. At time t , it observes an input x_t , a sky image with the recent irradiance history, and emits a forecast $\hat{I}_{t+H} = f_{\theta_t}(x_t)$ at horizon H with the parameters θ_t in use at time t . The measurement I_{t+H} that supervises this forecast arrives only after H has elapsed, which is the delayed-supervision regime of Sec. 2.4.3. Until then, the device retains a supervision key, the stored prediction, and a compact update state u_t . A detached embedding z_t with the auxiliary history suffices for a head-only update on a frozen encoder, whereas an encoder update needs the actual input. The three criteria of the adaptation

Tab. 4.1. Evaluation protocol of the three adaptation studies. The monitor of Sec. 4.3 is evaluated in series with the policy of Sec. 4.2; the execution path of Sec. 4.4 is evaluated on its own workload.

Study and criterion	Workload, streams, and platforms	Task metrics	System metrics	Replication
Conditional updates (Sec. 4.2); update justification	15-minute irradiance forecasting; Folsom 2016 replay, Poznań and Karpicko campaigns; Raspberry Pi 5 and Jetson Orin Nano	MAE, RMSE, and forecast skill; day-block bootstrap contrasts; recorded update actions	Per-action latency, energy net of idle, and peak process memory; update compute per day	One run per variant; few-shot three seeds
Input admissibility (Sec. 4.3); input admissibility	Folsom 2016 replay with synthetic sensing faults, in series with the policy above	Detection rate at the calibrated threshold and AUROC; forecast skill on faulty and fault-free frames; updates made and frames withheld	None on the target	One run; one draw per fault schedule
Accelerator execution (Sec. 4.4); update affordability	Head-only training of four backbones on CIFAR-100 and Oxford-IIIT Pet; Raspberry Pi 5 with Hailo-8L, Raspberry Pi 5 alone, Jetson Orin Nano	Test accuracy after five epochs	Step latency and energy, throughput against batch size, time to accuracy	Single seed

protocol become decidable at different times. Input admissibility is decided at t , when the frame arrives. Update justification is decided at $t + H$, when the label arrives. Update affordability is decided before deployment, from the platform and the parameterization. The chapter orders its sections by evidence instead. The update policy comes first (Sec. 4.2), because the monitor of Sec. 4.3 is evaluated in series with it, and the execution path comes last (Sec. 4.4), because it is evaluated on a separate workload.

4.1.2 Encoder–head factorization and design for stability

The forecaster is the composition of Eq. 2.2, drawn with its secondary uses in Fig. 2.1. The encoder produces the embedding $z_t = g(x_t; \theta_b) \in \mathbb{R}^d$, and the head consumes it together with the auxiliary conditioning r_t , the four irradiance readings at t , $t - 5$, $t - 10$ and $t - 15$ minutes, giving $\hat{I}_{t+H} = h([z_t, r_t]; \theta_{h,t})$ with $[\cdot, \cdot]$ denoting concatenation. The head-only update is the most restrictive of the parameter-efficient schemes of Sec. 2.4.4 and fixes the trainable subspace of Eq. 2.4 to the head coordinates,

$$\mathcal{S} = \{0\}^{|\theta_b|} \times \mathbb{R}^{|\theta_h|} \implies \theta_{b,t+1} = \theta_{b,t}, \quad \theta_{h,t+1} = \theta_{h,t} - \gamma_t \eta_t \nabla_{\theta_h} \mathcal{L}(\theta_t; \mathcal{B}_t). \quad (4.1)$$

Tab. 4.2. Notation used throughout this chapter. Symbols carry the same meaning across the three studies, and the frozen-encoder specialization of θ_t is the factorization on which the cost model of Eq. 4.6 rests.

Symbol	Meaning
x_t	Input at time t , the sky image and auxiliary history
r_t	Auxiliary conditioning signal (recent irradiance history)
H	Forecast horizon, also the supervision lag
$\hat{I}_{t+H}, I_{t+H}$	Predicted and measured irradiance at $t + H$
$g(\cdot; \theta_b), \theta_b$	Encoder and its parameters (frozen for head-only updates)
$\theta_b^{(3:4)}$	Parameters of the last two encoder stages (layer3, layer4 of ResNet50)
$h(\cdot; \theta_h), \theta_{h,t}$	Forecasting head and its trainable parameters at time t
$\theta_t = (\theta_b, \theta_{h,t})$	Parameters at t in the frozen-encoder notation
$z_t = g(x_t; \theta_b)$	Encoder embedding
$\mathcal{S}_t$	Trainable subspace selected at time t
γ_t	Update trigger, $\gamma_t \in \{0, 1\}$
η_h, η_b	Head and encoder learning rates ($\eta_b = 0.1 \eta_h$ in the proposed action)
$\mathcal{B}_t$	Evidence buffer admissible for learning at time t
u_t	Compact update state buffered until the label arrives
s_t, s_{anom}	Exclusion score of the sensing-fault monitor and its threshold
L	Realized absolute forecast error once I_{t+H} is observed
$T_1 < T_2$	Error thresholds selecting the update action
<i>Variant names used in the tables and figures of this chapter</i>	
Proposed, Conditional reduced-step	The policy of Eq. 4.3 with the reduced-step partial-encoder action of Eq. 4.4 at $T_1/T_2 = 10/100$; labeled <i>Proposed</i> in figures and <i>Conditional, reduced-step</i> in tables
Proposed (head only), Conditional head only	The same policy with $T_2 \rightarrow \infty$, so that every triggered update stays in the head (Eq. 4.1); labeled <i>Proposed (head only)</i> in figures and <i>Conditional, head only</i> in tables

Because Π_S annihilates the encoder coordinates, backpropagation terminates at the embedding and no encoder activation is retained. This also allows the frozen encoder to run on an inference-only accelerator (Sec. 4.4) and a monitor to read its features without a second pass (Sec. 4.3). Encoder updates widen $\mathcal{S}$ into the last two encoder stages and cost correspondingly more, so the policy of Sec. 4.2 reserves them for large errors. The early stage read by the monitor stays outside

the trainable subspace. This chapter and the next hold the input fixed to the RGB image and the irradiance history and vary the adaptation and exchange mechanisms. The physics-informed channels of Sec. 3.4 return in one crossed check at the end of Sec. 4.2.2. Table 4.2 summarizes the symbols used throughout the chapter.

Design for stability. The update law of Eq. 2.4 acts on a non-linear model with 10^6 to 10^7 coordinates from a delayed and noisy error signal, so its stability is assessed empirically over month-scale streams. Four design choices limit the influence of any single sample, namely a dead zone on small residuals, a trainable subspace restricted to the head or to the head with the last two encoder stages, so that the reachable parameter set is a small affine subset of $\mathbb{R}^P$, an encoder step reduced to one tenth of the head learning rate, and gradient clipping with reduced optimizer moment decay. Section 4.2.1 gives their values, and Sec. 4.2.2 shows which of them matter.

Evaluation setup. The forecasting studies use the second, chronological Folsom partition of Sec. 2.7.2, which trains on 2014 and 2015 and replays 2016 as a stream. Two deployment campaigns at unseen sites complete the three streams (Table 2.4). Poznań contributes 213 scored days and Karpicko 298. Every method starts from the same checkpoint. No evaluated sample was seen in training, and absolute errors are not comparable with those of Ch. 3.

Task metrics are MAE, RMSE, and FS against Smart Persistence at $H = 15$ minutes. They are pooled over the daylight population of each site. The mask keeps samples whose clear-sky irradiance exceeds 1 W/m^2 . This yields 245,538 samples over 362 days at Folsom, 123,523 over 213 days at Poznań, and 179,858 over 298 days at Karpicko. System metrics follow Sec. 2.7.5 on the Raspberry Pi 5 and the Jetson Orin Nano. Table 4.1 summarizes the workloads, platforms, metrics, and replication of the three studies.

4.2 Error-triggered updates under lagged supervision

The shift a deployed forecaster meets can invalidate the model completely, as the transfer result of Sec. 3.4 showed, and absorbing that shift is the purpose of on-device specialization. Error alone is a poor trigger, because irradiance is volatile at the minute scale on which the label arrives, and a policy that updated on every residual would trigger on noise and move the parameters away from an operating point that is still valid. This section therefore develops an error-triggered policy in the sense of Sec. 2.4.5, with the dead zone that section identifies as the standard remedy for triggering on noise, which maps the realized lagged error onto a finite action set of no update, a head update, and an encoder update. The reduced-step instantiation of the encoder action is compared against published alternatives on

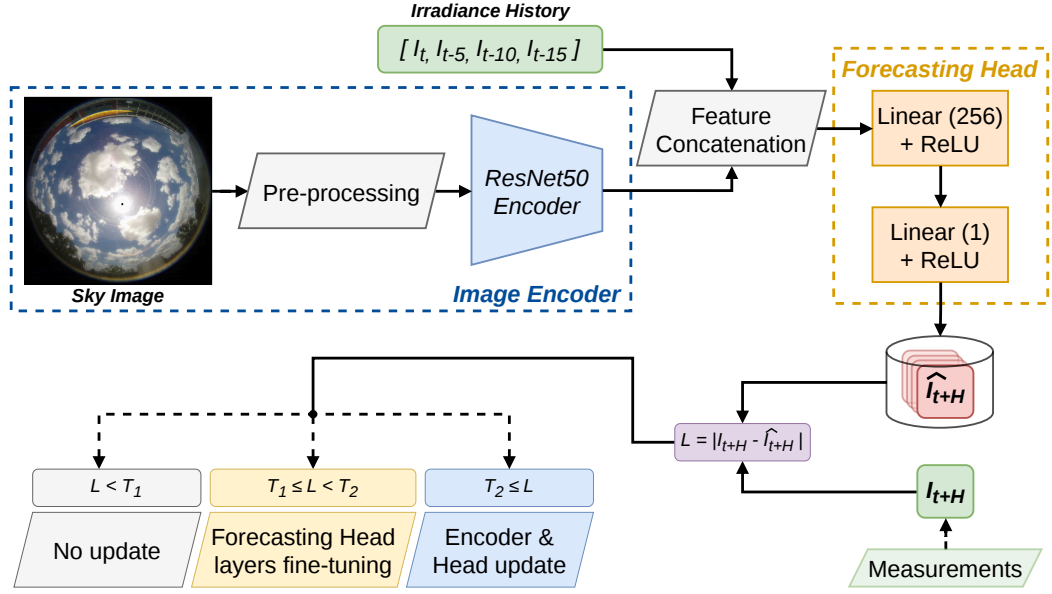


Fig. 4.1. Threshold-based conditional adaptation under lagged supervision. Each forecast is buffered with its update state until the measurement arrives after the horizon H ; the realized error then selects no update, a head update, or the reduced-step partial-encoder update of Eq. 4.4.

the three site streams, its per-action cost is measured on both target devices, and its daily update budget follows from the recorded actions.

4.2.1 Method

Conditional update rule. Once I_{t+H} becomes available, the absolute error of the stored forecast,

$$L = |I_{t+H} - \hat{I}_{t+H}|, \quad (4.2)$$

is mapped to an update action by two thresholds $T_1 < T_2$. The policy specializes the update law of Eq. 2.4 by making both the trigger and the trainable subspace functions of the realized error, with $\mathbb{K}[\cdot]$ the indicator function,

$$\gamma_t = \mathbb{K}[L \geq T_1], \quad \mathcal{S}_t = \begin{cases} \{0\}^P, & L < T_1, \\ \{0\}^{|\theta_b|} \times \mathbb{R}^{|\theta_h|}, & T_1 \leq L < T_2, \\ \mathcal{S}_{\text{enc}}, & L \geq T_2. \end{cases} \quad (4.3)$$

Small errors leave the model unmodified, intermediate errors select a head update, and large errors select an encoder update whose subspace $\mathcal{S}_{\text{enc}}$ is an instantiation choice. Since the per-step cost is governed by $\dim \mathcal{S}_t$, the expected update cost is a direct function of the error distribution and the thresholds. Algorithm 1 states the procedure and Fig. 4.1 depicts the lagged-label pipeline around it. The two thresholds are deployment choices fixed before installation. They are derived from the Standard Test Conditions reference irradiance of $1,000 \text{ W/m}^2$ [131], with T_1 and

Algorithm 1 Conditional on-device adaptation under lagged labels. When the monitor of Sec. 4.3 is active, it withholds flagged frames before this rule is applied.

Require: buffered tuple $(t, \hat{I}_{t+H}, u_t)$, lagged label I_{t+H} , thresholds $T_1 < T_2$

```

1:  $L \leftarrow |I_{t+H} - \hat{I}_{t+H}|$ 
2: if  $L < T_1$  then
3:   no update
4: else if  $T_1 \leq L < T_2$  then
5:   update the forecasting head (Eq. 4.1)
6: else
7:   reduced-step update of the last two encoder stages and the head (Eq. 4.4)
8: end if

```

T_2 set to 1 % and 10 % of that reference, 10 W/m² and 100 W/m². Anchoring them to a reference the deployment already possesses makes the setting reproducible across sites and independent of any of them. On the held-out validation days of the source partition, the two values sit at about the 61st and 92nd percentiles of the static model’s absolute error, so they approximate a rule set from source data alone; the pair that rule would give, the median and 90th percentile of the same distribution, 5.4 and 81.1 W/m², is evaluated alongside them in Sec. 4.2.2.

Encoder actions. The originating publication [17] specified $\mathcal{S}_{\text{enc}} = \mathbb{R}^P$, an update of the whole network at the head learning rate. The proposed instantiation changes it in two ways, for the reason Sec. 4.2.2 establishes. The trainable subspace is restricted to the last two encoder stages together with the head, and the encoder coordinates are stepped at one tenth of the head learning rate,

$$\mathcal{S}_{\text{enc}} = \{0\}^{|\theta_b| - |\theta_b^{(3:4)}|} \times \mathbb{R}^{|\theta_b^{(3:4)}|} \times \mathbb{R}^{|\theta_h|}, \quad \eta_b = 0.1 \eta_h. \quad (4.4)$$

A head-only policy, obtained with $T_2 \rightarrow \infty$ so that every triggered update stays in the head, serves as the reference for the proposed action, and the ablation of Sec. 4.2.2 adds the original full-encoder action and variants that restrict the subspace without reducing the step, add replay of buffered source samples, or add an anchor toward the deployed weights.

Comparators. The policy is compared with three families of alternatives that differ in what supervision they need and when they act. Instance-based test-time adaptation updates the model before prediction from the incoming sample alone. MEMO [55] and NOTE [56] were designed for class outputs and minimize prediction entropy, which has no counterpart for a scalar forecast, so both are reimplemented with a prediction-variance objective, the mean squared deviation of a set of forecasts from their mean, which rewards forecasts that agree with one another. The MEMO variant is episodic. For every frame, it resets the model, forms eight views with the geometric transforms of training, takes one Adam step at a learning rate of 10^{-4} on all parameters to make the eight forecasts agree, and forecasts from the

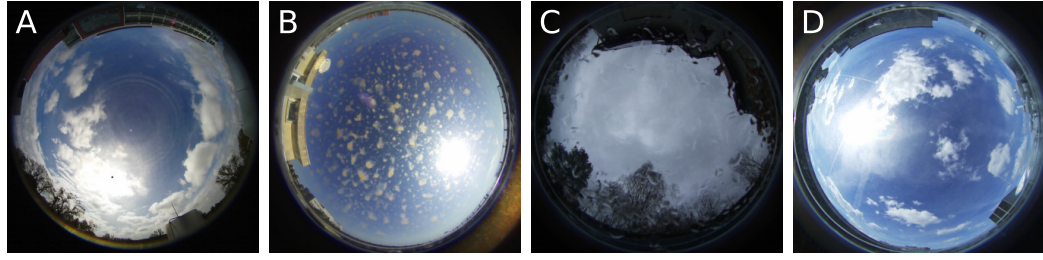


Fig. 4.2. Representative sky images from the training site and the deployment sites. Differences in optics, exposure, horizon obstruction, and prevailing cloud regime are visible directly. Sites: (A) Folsom, the training site; (B) Modena, the week-scale stream of the originating publication, shown for contrast; (C) Karpicko and (D) Poznań, the two deployment campaigns of Table 4.3.

unaugmented frame with the adapted weights. The NOTE variant is cumulative. It keeps a first-in-first-out buffer of the last 64 frames, takes one Adam step at the same learning rate on the batch-normalization affine parameters after every 64 frames, tracking the normalization statistics at momentum 0.1, and forecasts every frame with the current weights. Few-shot site adaptation performs a single adaptation per site from a small local dataset collected before deployment, with no subsequent loop [132], and is evaluated with one and seven days of local data. Delayed-supervision online adaptation defers updates until measurements arrive, as the present policy does. Zhang et al. [74] adapt only the final layer through a Sherman–Morrison recursive least-squares update, and Lara-Benítez et al. [75] perform sliding-window mini-batch gradient adaptation over a rolling buffer of the most recent labeled samples. Smart Persistence, defined in Eq. 2.10, is the statistical reference at every site.

Optimization and streams. The deployed checkpoint, including its forecasting head and its offline BerHu training recipe, is specified in Table 2.5. The online phase minimizes the squared error and performs one optimizer step per buffered sample once its label arrives, at a head learning rate $\eta_h = 10^{-5}$ with weight decay 10^{-4} , moment decay rates of 0.5 and 0.975, and gradient norm clipping at a maximum norm of 1.0, the safeguards of Sec. 4.1.2. Figure 4.2 shows the site differences that keep a model fitted at one location from transferring unchanged to another; the Karpicko and Poznań campaigns replace the week-scale Modena comparison of the originating publication.

4.2.2 Evaluation

Comparative benchmark. Table 4.3 positions the conditional update policy against the alternatives over the campaign period of each site, Fig. 4.3 gives the paired contrasts with their intervals, and Fig. 4.4 follows the arms month by month. Four patterns follow. First, deployment shift can erase the learned model’s advantage over the local statistical reference. The frozen forecaster retains 10.73 % forecast skill at

Tab. 4.3. Recorded-stream comparison at three sites. MAE and RMSE are pooled errors in W/m^2 and lower is better. Higher FS (%) against the same-site Smart Persistence RMSE is better. Few-shot rows average three seeds and all others are single runs. Bold marks the best value per column within each site, excluding Smart Persistence. MEMO and NOTE denote the regression implementations of Sec. 4.2.1. RLS is final-layer recursive least squares of Zhang et al. [74]. Sliding window is the adaptation of Lara-Benítez et al. [75]. Few-shot uses one or seven days of local data following Paletta et al. [132]. Conditional denotes the head-only and reduced-step variants of Eqs. 4.3 and 4.4.

Site	Method	MAE ↓	RMSE ↓	FS (%)↑
Folsom	Smart Persistence	35.95	86.10	0.00
	Static model	31.51	72.60	15.68
	MEMO	33.26	72.74	15.51
	NOTE	104.79	138.62	−61.01
	RLS Zhang et al. [74]	31.31	71.94	16.44
	Sliding window Lara-Benítez et al. [75]	31.50	69.48	19.30
	Few-shot Paletta et al. [132], 1 day	31.78	73.15	15.04
	Few-shot Paletta et al. [132], 7 days	37.14	74.75	13.18
	Conditional, head only	31.03	70.16	18.51
	Conditional, reduced-step	31.16	69.54	19.23
Poznań	Smart Persistence	67.73	138.99	0.00
	Static model	69.56	124.08	10.73
	MEMO	68.00	122.48	11.88
	NOTE	96.66	135.68	2.38
	RLS Zhang et al. [74]	66.56	120.84	13.06
	Sliding window Lara-Benítez et al. [75]	58.92	111.78	19.57
	Few-shot Paletta et al. [132], 1 day	69.19	124.44	10.46
	Few-shot Paletta et al. [132], 7 days	63.49	121.50	12.58
	Conditional, head only	59.65	114.85	17.37
	Conditional, reduced-step	58.58	112.70	18.91
Karpicko	Smart Persistence	67.19	142.46	0.00
	Static model	80.40	140.14	1.63
	MEMO	79.58	140.90	1.09
	NOTE	134.07	169.51	−18.99
	RLS Zhang et al. [74]	75.84	134.54	5.56
	Sliding window Lara-Benítez et al. [75]	66.76	124.77	12.42
	Few-shot Paletta et al. [132], 1 day	68.65	136.60	4.11
	Few-shot Paletta et al. [132], 7 days	68.53	138.38	2.87
	Conditional, head only	66.58	127.99	10.16
	Conditional, reduced-step	65.90	124.92	12.32

Poznań but only 1.63 % at Karpicko, and Smart Persistence has the lower MAE at both sites (67.73 against 69.56 and 67.19 against 80.40 W/m^2), so the transferred model trades a worse typical error for smaller extreme errors. Second, adaptation helps most where there is shift to absorb. On the held-out Folsom year, the head-only and the reduced-step instantiation have the lowest MAE of any method (31.03 and 31.16 versus 31.51 for the static model), and the reduced-step instantiation raises forecast skill by 3.5 points over the static model. At the unseen sites, it reduces MAE

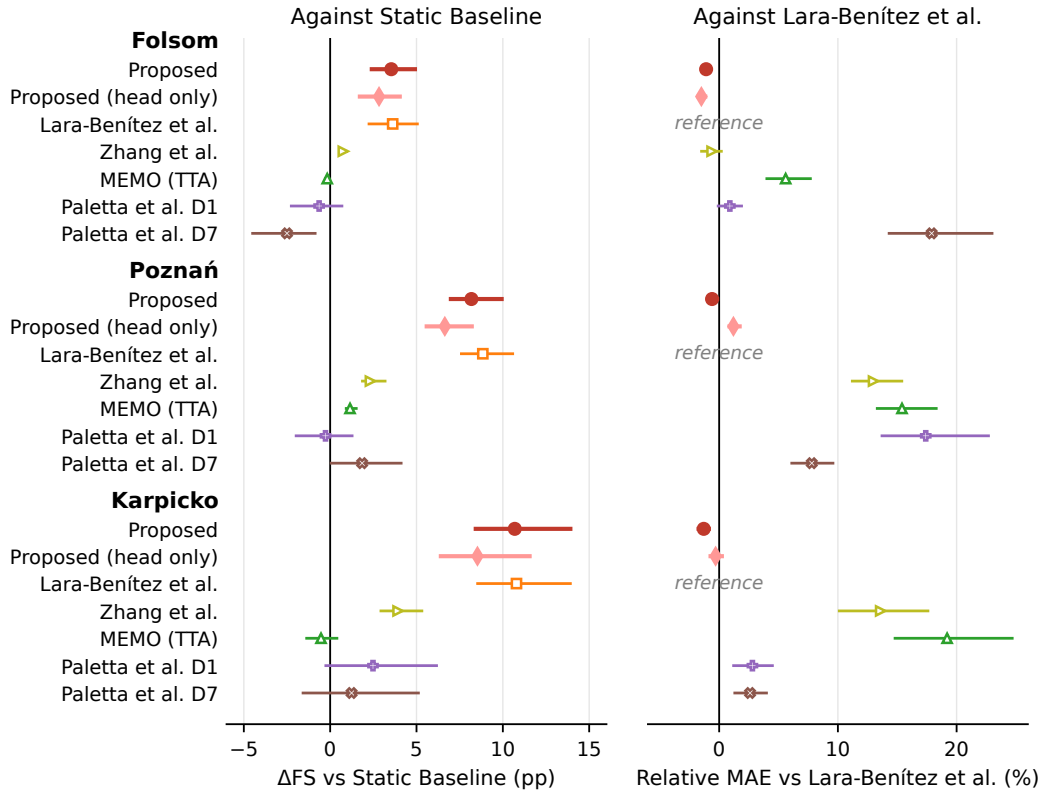


Fig. 4.3. Paired contrasts of each adapted method against the static model (left, Δ FS in percentage points, positive favors the method) and against the sliding-window adaptation of Lara-Benítez et al. [75] (right, MAE as a percentage of that method's MAE minus 100). Intervals are moving-block bootstraps over days (block 7, 2000 date-paired resamples), 95 % for Δ FS and 90 % for relative MAE; values beyond the axis are printed. Single runs, so the intervals describe temporal variability, unadjusted for the 21 contrasts shown.

by 15.8 % at Poznań and 18.0 % at Karpicko, with skill gains of 8.2 and 10.7 points and intervals well clear of zero; the gain is present in every month at Karpicko, largest from November to February, and in every month but December at Poznań, where all adapted methods and Smart Persistence deteriorate together.

Third, the two test-time adaptation methods do not help. The MEMO variant changes the error only marginally, and the NOTE variant degrades it substantially, to $3.3\times$, $1.4\times$ and $1.7\times$ the static MAE at the three sites, which follows from its objective; minimizing the variance of forecasts across 64 different frames rewards a constant forecast. Fourth, the reduced-step variant has 0.6 % to 1.3 % lower MAE than the sliding-window adaptation of Lara-Benítez et al. [75] at all three sites, with 90 % intervals a few percent wide. Its forecast skill is 0.07 and 0.10 points lower at Folsom and Karpicko, differences the intervals do not resolve, and 0.66 points lower at Poznań. The head-only instantiation trails on forecast skill by 0.8 to 2.3 points at every site, so the 0.7 to 2.2-point increment of the reduced-step action over head-only updates must be justified by the update-cost measurements below.

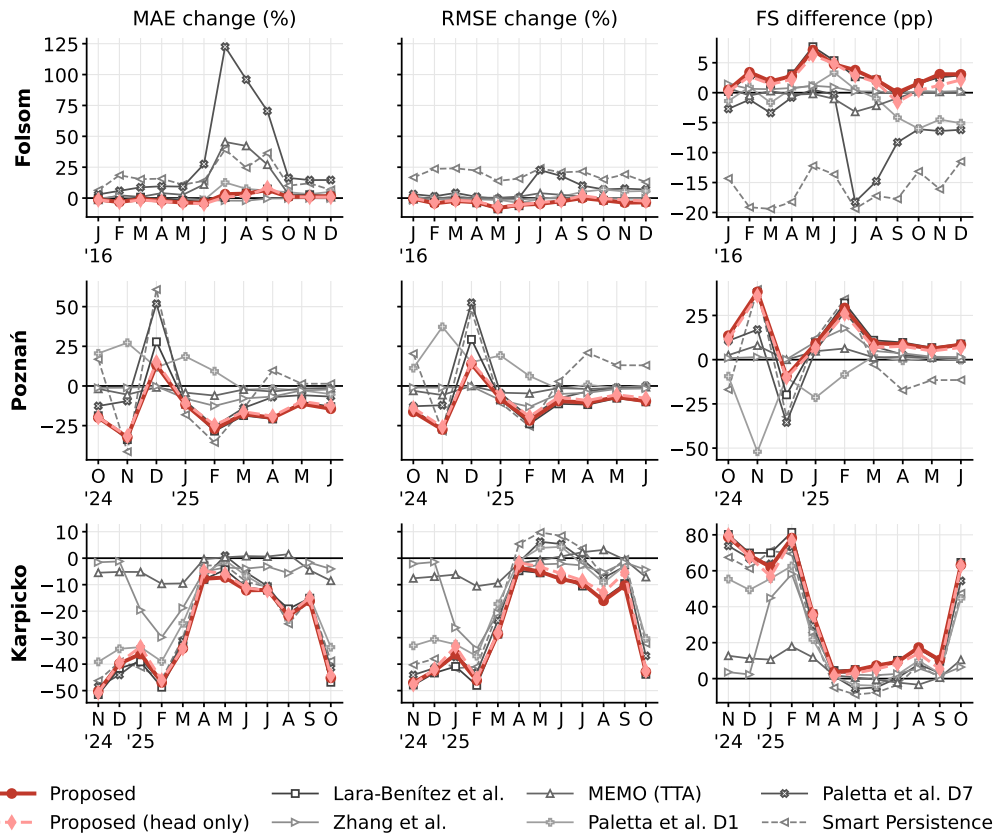


Fig. 4.4. Monthly error and forecast skill relative to the static model. MAE and RMSE in percent, forecast skill in percentage points. Negative error changes and positive forecast skill differences are better; zero is the static reference. Proposed is the reduced-step action of Eq. 4.4, the dashed Proposed line is head-only, and the comparators carry the author names of Table 4.3. NOTE is omitted because its errors dominate the plotted range.

The encoder actions have different failure modes. The first full-network update at the head learning rate has MAE 32 % above the static model at Folsom and forecast skill 5.4 points below it, whereas the reduced-step partial-encoder configuration is 1.1 % better on MAE and 3.5 points better on forecast skill. The full-encoder policy deteriorates progressively after May, reaching about 140 % higher monthly MAE in July and September without recovering during the record, while at the unseen sites its campaign MAE stays within 3 % of the reduced-step configuration. Table 4.4 separates the two changes. Restricting updates to the last two stages alone leaves a worst-month increase of 102 %, and adding replay leaves 92 %; adding the tenfold smaller encoder learning rate to the restricted configuration reduces the increase to 7 %, the level of the anchor variant and of the head-only reference (8 %). Restriction alone therefore did not prevent the deterioration, and the tested combination of restriction and learning rate reduction did.

Sensitivity to the thresholds. The lower block of Table 4.4 varies each threshold once around the reference pair and adds the pair set from the source validation

Tab. 4.4. Ablation of the high-error action of Eq. 4.3 and sensitivity to its thresholds. Upper block, the encoder action at the reference thresholds $T_1/T_2 = 10/100$ W/m²; lower block, the reduced-step action at other thresholds, the last pair being the median and 90th percentile of the static model’s absolute error on the source validation days. FS (%) per site and mean MAE over the three sites (W/m²); at Folsom, the worst monthly MAE relative to the static model and the number of months with higher MAE than the static model; update compute per day on Raspberry Pi 5 from the recorded action counts and the measured latency of each variant’s own update action (Table 4.5), mean over the three sites; the replay variant’s extra mini-batch is not timed, so its value is a lower limit. Replay draws buffered source samples into each update; anchor adds a draw toward the deployed weights. Single runs on the evaluation streams.

Configuration	FS Folsom	FS Poznań	FS Karpicko	Mean MAE	Worst month (%)	Months worse	RPi 5 (s/day)
<i>Encoder action, $T_1/T_2 = 10/100$</i>							
full encoder	10.32	16.99	10.11	57.63	141	12/12	128
layer3+4	15.42	19.89	13.46	54.47	102	12/12	109
layer3+4, replay	13.71	19.37	14.05	54.76	92	12/12	109
layer3+4, anchor	17.82	17.80	8.24	52.88	8	7/12	100
reduced-step (proposed)	19.23	18.91	12.32	51.88	7	6/12	100
head only	18.51	17.37	10.16	52.42	8	6/12	51
<i>Thresholds T_1/T_2 (W/m²), reduced-step action</i>							
5 / 100	19.24	18.96	12.34	51.52	1	3/12	111
20 / 100	19.07	18.79	12.27	53.23	41	8/12	88
10 / 50	19.30	19.20	12.91	50.96	6	6/12	135
10 / 200	19.01	18.68	11.52	52.77	7	6/12	74
5.4 / 81.1 (validation p50 / p90)	19.28	19.05	12.57	51.25	1	3/12	119

days. Every variant keeps the gain over the static model, and the spread of forecast skill across the tested range is 0.3 points at Folsom, 0.5 at Poznań and 1.4 at Karpicko, growing with the shift from the source site. The encoder threshold is the consequential one. Halving T_2 raises skill by up to 0.6 points for about a third more update compute, doubling it costs up to 0.8 points for a quarter less, whereas moving T_1 changes skill by at most 0.16 points; the single marked monthly deterioration appears for $T_1 = 20$ W/m² at Folsom, where the wider dead zone leaves errors uncorrected for longer. The validation-derived pair scores 0.05 to 0.25 points above the reference at each site for a fifth more compute, a margin within the run-to-run variability that the single-run design leaves unmeasured, so the rule set from the Standard Test Conditions reference is as good as what source data alone would select. Within the tested range, the thresholds set an FS–compute trade-off, not a fragile optimum.

Update actions and device cost. Table 4.5 reports the actions the policy selected. The reduced-step instantiation triggers an encoder update on 8.6 % of buffered samples at Folsom and on 17 % to 19 % at the unseen sites, and leaves the model

Tab. 4.5. Recorded update actions per site for the two variants of the policy, as a share of buffered samples (%), and the update compute per day obtained by multiplying the recorded head and encoder counts by the per-action latencies listed below the table. The budget assumes the timed latency for every triggered update and no overlap with inference.

Encoder action	Site	Action share (%)			Update compute (s/day)	
		none	head	encoder	RPi 5	Orin Nano
reduced-step (proposed)	Folsom	56.9	34.4	8.6	73	13.2
	Poznań	34.5	48.5	17.0	106	18.8
	Karpicko	30.3	50.5	19.2	121	21.4
none (head only)	Folsom	57.3	42.7	0.0	41	8.4
	Poznań	34.8	65.2	0.0	54	11.0
	Karpicko	31.0	69.0	0.0	59	12.1

Per-action latency in ms per sample, inference / head update / reduced-step update / full-model update, mean $\pm$ standard deviation over 200 repetitions on synthetic inputs of the deployed shape, one process per method running the update code and hyperparameters of its campaign run. Raspberry Pi 5, 133.6 ± 0.9 / 146.0 ± 1.5 / 685.3 ± 19.6 / 823.9 ± 28.5 . Jetson Orin Nano, 21.6 ± 0.5 / 29.8 ± 0.2 / 110.8 ± 3.7 / 133.6 ± 4.4 . The inference entry times the reduced-input forecaster inside the adaptation process and differs from the inference stage of Table 3.14, which times the seven-channel forecaster in a separate pipeline.

unmodified on 57 % of samples at Folsom compared with 30 % to 35 % at the unseen sites. This is the intended behavior of a dead zone. Updates are more frequent where the deployed prior is wrong. Multiplying the recorded counts by the measured per-action latencies gives the update compute per day, 73 s to 121 s on the Raspberry Pi 5 and 13 s to 21 s on the Orin Nano, compared with 41 s to 59 s and 8 s to 12 s for the head-only variant, so the additional forecast skill of the reduced-step action costs 1.8 to $2.1 \times$ the head-only update time on the Pi and 1.6 to $1.8 \times$ on the Orin. A head update costs 1.1 inferences on the Pi and 1.4 on the Orin, a reduced-step update 5.1 on both and a full-model update 6.2, so restricting the encoder action to the last two stages saves about a sixth of the full-model step. The timed head update recomputes the encoder forward pass so that a cached-embedding path would cost less still.

Table 4.6 places the policy beside the comparators on the two devices. Per update event, the reduced-step action is the most expensive gradient step after the two test-time adaptation methods, about 2 times the sliding-window step of Lara-Benítez et al. [75] on the Pi and 3.7 times on the Orin, but it updates on 9 % to 19 % of samples while the comparator updates on every one. Per frame, inference included, the reduced-step policy therefore uses 27 % to 48 % less time and energy than sliding-window adaptation on the Pi at the three sites. On the Orin, a four-sample batch step costs no more than a head update, so the policy uses 19 % to 44 % less energy but 6 % to 12 % more time at the two unseen sites, where the encoder action fires most. All strategies fit the one-minute sampling interval, with

Tab. 4.6. Per-event and per-frame cost of the adaptation strategies on the two target devices: the update event in ms; the stream cost per frame, inference plus the update weighted by how often the method updates, in ms and mJ; and peak process memory (resident set size (RSS)) in MB. Processes, inputs, and repetition count as in the note to Table 4.5. Energy is board power sampled during the timed block, from the on-board PMIC on Raspberry Pi 5 and from the tegrastats module input on Orin Nano in the MAXN power mode with `jetson_clocks` enabled, net of the idle power measured before the run, 1.87 W and 3.43 W; module telemetry excludes carrier-board losses, so the two devices are not compared with each other. The per-frame values follow from the measured event costs and the recorded update rates, every sample for the comparators, one batch step per 64 samples for NOTE, and the action shares of Table 4.5 for the conditional policies (Folsom / Poznań / Karpicko). No network or concurrent workload.

Method	Update (ms)	Per frame (ms)	Per frame (mJ)	Peak RSS (MB)
<i>Raspberry Pi 5, CPU, four threads</i>				
Static model	–	134	541	1040
Conditional, head only	146	195 / 228 / 234	755 / 888 / 910	1041
Conditional, reduced-step (proposed)	685	244 / 322 / 340	913 / 1185 / 1246	1086
Lara-Benítez et al. [75]	333	468	1717	1041
Zhang et al. [74]	136	268	1067	1072
MEMO [55]	2864	2998	8329	1437
NOTE [56]	17293	399	1234	3138
<i>Jetson Orin Nano, GPU</i>				
Static model	–	22	90	1604
Conditional, head only	30	35 / 41 / 42	138 / 164 / 168	1838
Conditional, reduced-step (proposed)	111	42 / 55 / 58	181 / 247 / 262	2008
Lara-Benítez et al. [75]	30	52	324	1838
Zhang et al. [74]	24	46	186	1703
MEMO [55]	144	167	1705	2296
NOTE [56]	424	31	203	3957

MEMO at 22 inferences per frame on the Pi, so the differences are idle time and energy. The reduced-step policy occupies the Pi for 244 ms to 340 ms per frame, 0.4 % to 0.6 % of the acquisition period, well inside the slack that Table 3.14 left after inference, and over a Folsom-length day of 678 frames it draws 0.6 kJ to 0.85 kJ above idle on the Pi, 0.4 % to 0.5 % of the board’s idle energy over 24 hours, and 0.12 kJ to 0.18 kJ on the Orin. Peak memory differs little between the methods. The conditional, sliding-window and recursive-least-squares processes peak within 5 % of the static-inference process on the Pi (1,040 MB to 1,086 MB) and within about 25 % on the Orin, where the difference is accelerator workspace. Only MEMO (1.4 GB to 2.3 GB) and NOTE (3.1 GB to 4.0 GB) differ markedly, the latter through its 64-sample buffer step.

Tab. 4.7. Storage duty implied by the forecast errors of each method on the scored population, per megawatt of plant, under the forecast-following proxy of Sec. 3.4.2: energy cycled through the battery per scored day (kWh), the 99th percentile of battery power (kW), the 95th percentile over days of the intraday range of the cumulated deviation (kWh), and the mean error (bias, kW). One-minute frames, round-trip efficiency 0.92, no state-of-charge limit, price signal or degradation model; percentages are relative to the static model at the same site. Smart Persistence is recomputed per frame on the same population. Single runs.

Method	Cycled per day (kW h, %)		Battery power p99 (kW, %)		Intraday range p95 (kW h, %)		Bias (kW)
<i>Folsom</i>							
Smart Persistence	442	14	416	25	169	−57	0.6
Static	387	−	333	−	388	−	−4.6
Conditional, reduced-step	383	−1	315	−5	274	−29	−0.3
<i>Poznań</i>							
Smart Persistence	712	−3	633	29	695	−2	−23.5
Static	731	−	492	−	711	−	2.3
Conditional, reduced-step	615	−16	468	−5	488	−31	−1.4
<i>Karpicko</i>							
Smart Persistence	735	−16	668	17	441	−66	−15.6
Static	879	−	569	−	1,305	−	−10.2
Conditional, reduced-step	721	−18	529	−7	479	−63	−0.6

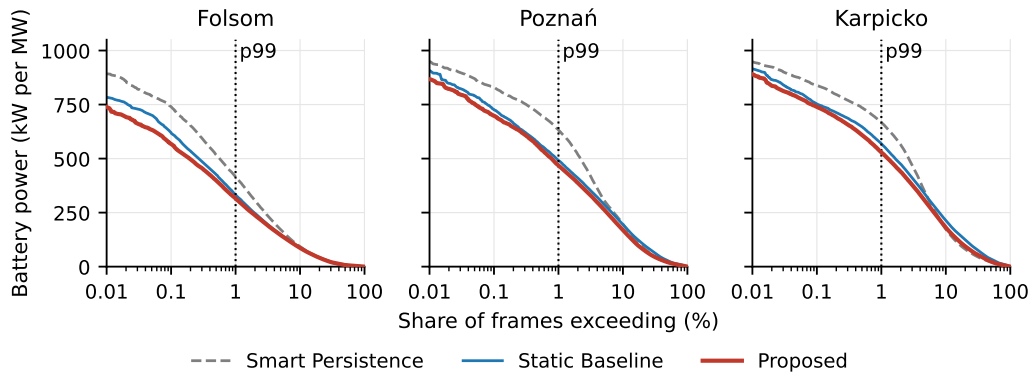


Fig. 4.5. Battery-power duration curves per site: the absolute forecast error of every scored frame, as battery power per megawatt of plant, sorted in decreasing order against the share of frames that exceed it (logarithmic axis). The value at 1 % is the 99th percentile of Table 4.7; the left end is the largest error of the year. A curve below another needs less battery power at that share of frames; a crossing, as between Smart Persistence and the static model at Poznań and Karpicko, means one method has the larger extreme errors and the other the larger typical ones.

Storage duty of the forecast errors. The forecast-following proxy of Sec. 3.4.2 translates the errors into storage duty for a 1 MW plant that schedules its output on the 15-minute forecast and absorbs the deviation in a battery (Table 4.7). The energy cycled per day follows the absolute error and ranks the arms as MAE does,

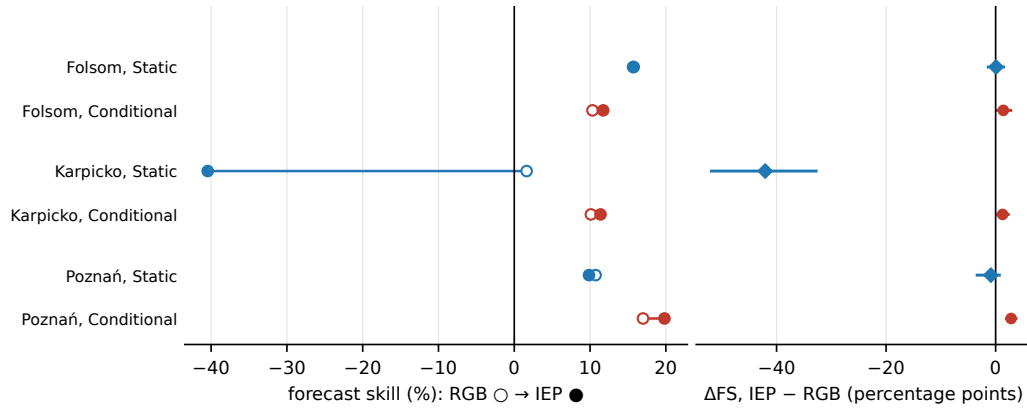


Fig. 4.6. Forecast skill of the static model and of the conditional policy with the reduced input of this chapter (RGB, hollow) and with the seven-channel physics-informed input of Sec. 3.4 (IEP, filled), on the populations of Table 4.3 (left), and the IEP-minus-RGB difference with its 95 % day-block bootstrap interval (right). Conditional rows use the policy as first specified in the originating publication; one run per cell.

the 99th percentile of battery power follows the largest errors as RMSE does, and the intraday range of the cumulated deviation, the energy the battery must hold within a day, follows persistent error. The reduced-step policy is the only variant that lowers all three at every site relative to the static model, cycling by 1 % at Folsom and 16 % to 18 % at the unseen sites, the power percentile by 5 % to 7 %, and the intraday energy range by 29 % to 63 %, the last because it removes the bias of the static model, 2 kW to 10 kW per megawatt before and below 1.5 kW after adaptation. Smart Persistence needs 17 % to 29 % more battery power than the static model at every site while cycling less at the Polish sites, and Fig. 4.5 shows why, since its duration curve crosses the static model’s near the largest 5 % to 10 % of errors. The proxy has no state-of-charge limit, price signal, or degradation model and a different time step from Ch. 3, so only relative changes carry across chapters.

Physics-informed input under adaptation. A second forecaster with the seven-channel input of Sec. 3.4, trained on the same partition and with the sun-mask camera model refitted for each site, was run static and under the policy as first specified, one run per cell (Fig. 4.6). Two observations carry over. The static physics-informed model collapses at Karpicko (forecast skill -40.5%), because from May to August trees occlude the noon Sun in that camera’s field of view while the pyranometer is unobstructed, so the sun-position channel and the image disagree and the model underpredicts by up to 290 W/m^2 at noon; blanking the image channels on the affected days lowers its RMSE from 243 to 173 W/m^2 . Adaptation recovers it, and the adapted physics-informed model is the best cell at every site, by 1.3 to 2.8 forecast skill points over the adapted reduced-input model. The two checkpoints differ in training augmentation as well as input, so the check shows that the physics-

informed forecaster benefits from adaptation without isolating the contribution of the channels.

4.2.3 Conclusions

Across recorded streams of 213 to 362 scored days per site, the reduced-step partial-encoder policy improves on the static checkpoint at every site, by 3.5 forecast skill points at Folsom and by 8.2 and 10.7 points at the two unseen sites. It has 0.6 % to 1.3 % lower campaign MAE than sliding-window adaptation at a per-frame energy 19 % to 48 % lower on both devices. Its two changes to the original action are not interchangeable, since restricting the trainable stages alone left the source-site deterioration in place and reducing the encoder step removed it. The thresholds set an FS–compute trade-off within the tested range, the encoder threshold being the consequential one, and the pair fixed from the Standard Test Conditions reference performs within 0.3 points of the pair a site would fit from its own validation errors. In the storage proxy, the policy lowers cycling, peak battery power, and the intraday energy range at every site. The recorded actions and the measured per-action costs give 73 s to 121 s of daily update compute on the Raspberry Pi 5 and 13 s to 21 s on the Orin Nano, inside the idle time that the acquisition period leaves.

Limitations.

- The decay factor, the trainable stages, and the choice among the ablation variants were selected on the same recorded streams used for reporting. Every method except few-shot adaptation is one run, so the intervals describe temporal variability over days and the 0.7 to 2.2-point gains of the reduced-step action over head-only updates are the results most exposed to this.
- Retention is not measured. The reported gains describe the deployment stream alone, and forgetting of the source distribution over returning seasons is outside the question this chapter asks.

4.3 Input admissibility, detecting faults and testing exclusion

The policy of Sec. 4.2 acts on buffered samples once their labels arrive, and in Sec. 4.2 every sample reached it. Not every departure from the training distribution is a fault, and not every fault is unlearnable. Dirt, insects, and birds on the dome, water on it, and acquisition faults in the camera all change the frame without changing the irradiance it must predict. This section tests whether image-based fault detection justifies excluding such frames from learning. It compares the pooled-embedding screen of the originating publication [16] with a per-position score that detects selected faults, and then evaluates exclusion by its consequence, the forecasts of

the policy with and without the flag on clean and corrupted replays of the Folsom year.

4.3.1 Method

Data and calibration. All detectors are fitted on the Folsom record in the second partition of Table 2.4. Every fifth frame of 2014 and 2015 is used and split by whole days, so that 458 days (62,251 frames) fit each detector and 115 held-out days (15,466 frames) set its threshold at the 99th percentile of the detector’s own calibration scores, a nominal exceedance rate of 1 % following Sec. 2.3.3. The held-out 2016 year is replayed as a stream of unobserved frames. Clean denotes the unmodified stream; because its real faults are not annotated, the rate at which a detector flags it is a nominal-frame flag rate.

Pooled and per-position scores. The pooled screen maps each frame to $z_t = g(x_t; \theta_b) \in \mathbb{R}^{2,048}$ and scores it with an Isolation Forest [53] of 100 estimators fitted to nominal training embeddings; a Mahalanobis distance and a k -nearest-neighbor distance on the same embeddings, and two model-free references on raw pixels, a dark-pixel fraction and a Laplacian-variance blur score, are compared with it. Scoring traverses the fitted trees without a second encoder pass. The fixed camera motivates a separate nominal feature distribution at each image position. The per-position detector reads the encoder’s second stage, a $512 \times 16 \times 16$ tensor at the 128×128 input, one cell per 8×8 pixel block, pools each cell with its 3×3 neighborhood and projects it to 128 dimensions with a fixed random matrix. For each of the 172 cells whose block lies inside the circular field of view, a memory bank holds that cell’s features from 2,000 nominal frames, and a cell’s score is its nearest-neighbor distance to its own bank divided by the 95th percentile of the same distance on 1,000 other nominal frames. The design adapts the nominal patch-feature memory bank of PatchCore [133] to a fixed camera through position-specific banks and distance normalization. The bank occupies 176 MB in FP32, and because the second-stage weights lie outside the trainable subspace of Eq. 4.4, one bank stays valid while the policy adapts.

The exclusion flag and the maintenance alert. A frame score is a statistic of the 172 cell scores, and the choice of statistic decides which faults are visible. The maximum over all cells is dominated by the outermost ring, where the low Sun and its reflections sit. On the clean 2016 stream, 95 % of the frames it flags are due to that ring, and three quarters of them fall at solar elevations below 20° or within two cells of the projected Sun. The retained flag therefore uses two statistics for two footprints. The global score G is the median over all cells and responds to frame-wide departures, an acquisition fault, or water over the dome. The core score C is the maximum over the 72 cells at least three cells from the border and responds

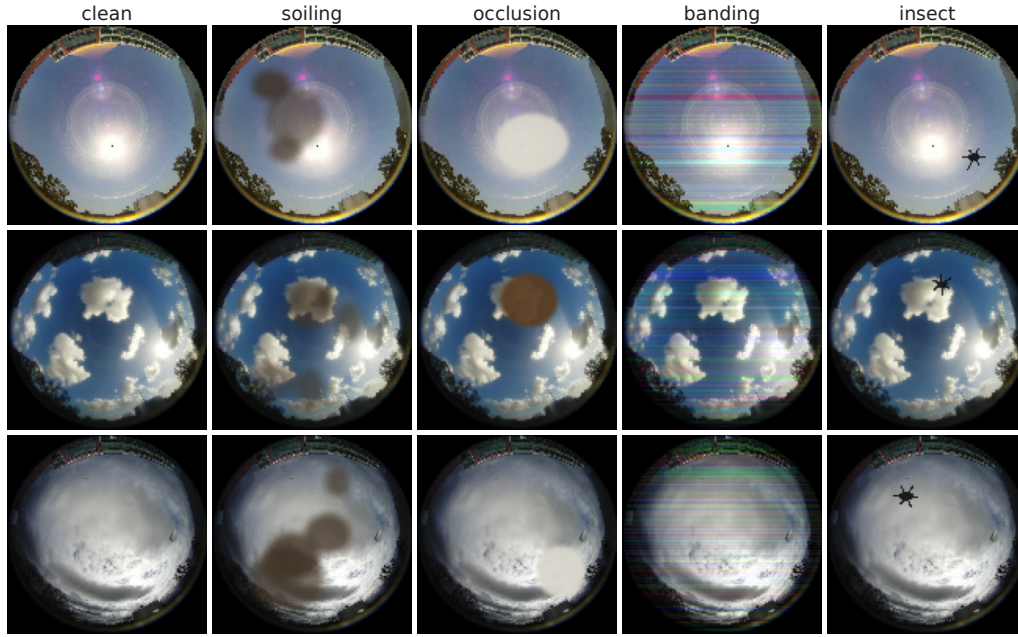


Fig. 4.7. Synthetic sensing-path faults rendered onto three real Folsom 2016 frames (rows: clear, cumulus, overcast). Columns: the clean frame; soiling; occlusion, whose color and shape are drawn per episode; the same two faults at the rim of the field of view; banding, an acquisition fault; and an insect on the dome. Every frame carries its valid irradiance label; only the image is altered.

to compact objects away from the rim. The exclusion score is $s_t = \max(G/t_G, C/t_C)$, with t_G and t_C the 99.5th percentiles of G and C on the calibration frames, so that $s_t > s_{\text{anom}} = 1$ flags about 1 % of nominal frames. A flagged frame sets $\gamma_t = 0$ in Eq. 4.3 and is withheld from the update loop, while its forecast is issued as usual. The rule is framewise, so persistent and transient exceedances are treated alike. A persistent fault can also call for maintenance, so an alert is evaluated separately. The alert is a causal rolling median of a frame score over the preceding 30 or 60 minutes of the same day, thresholded at its 99th percentile on 30 whole calibration days. The criterion is that every soiling episode raises an alert within two hours while alerts are raised on at most 1 % of fault-free frames.

Synthetic faults and streams. No annotated sensing faults exist for any of the three sites, so detection is measured on faults rendered onto clean 2016 frames (Fig. 4.7). Soiling is three to six diffuse dark patches persisting for one to three whole days; occlusion is an opaque object of a color and shape drawn per episode, standing in for a bird and lasting 5 to 30 minutes; banding is horizontal gain bands and row noise redrawn every frame, modeled on an acquisition fault of the Folsom camera and lasting 1 to 5 minutes; and insect is a body of a few pixels with legs, lasting 2 to 15 minutes. Soiling and occlusion are also rendered at the rim of the field of view, where the flag discards evidence by construction. The policy is then run on four streams. These are the clean year, a year with mixed episodes of every fault on

Tab. 4.8. Detection on 3,070 Folsom 2016 daylight frames per condition. (a) Percentage flagged at each detector’s calibrated threshold; the nominal column is the flag rate on unmodified frames. Calibration targets a 1 % exceedance rate (0.5 % per exclusion statistic). (b) AUROC against the nominal frames. Higher fault detection and AUROC are better. Rim faults are centered in the outer fifth of the field-of-view radius. Detector definitions are in Sec. 4.3.1.

(a) Flagged frames (%)							
Detector	Nominal	Soil.	Soil. rim	Occl.	Occl. rim	Band.	Insect
Isolation Forest (pooled)	1.2	0.9	1.2	1.2	1.4	1.7	1.1
Dark-pixel fraction	0.7	0.6	0.1	22.9	13.7	0.6	1.1
Per-position maximum	1.8	13.3	9.2	50.5	32.1	58.3	10.8
Exclusion flag	1.2	39.8	8.5	79.2	35.9	85.4	33.0

(b) AUROC against nominal frames					
Detector	Soiling	Occlusion	Banding	Insect	
Isolation Forest (pooled)	0.69	0.62	0.77	0.54	
Dark-pixel fraction	0.49	0.74	0.49	0.52	
Per-position maximum	0.89	0.96	0.97	0.84	
Exclusion flag	0.96	0.99	0.99	0.94	

3.9 % of daylight frames, a continuous soiling from 1 July to 15 August followed by cleaning, and a permanent occluder from 1 September to the end of the year. Three further mixed streams have 1, 5 and 15 % of frames faulty. Each configuration is one run from the checkpoint of Sec. 4.2.1 on the population of Table 4.3, and each fault stream is one draw of its schedule.

4.3.2 Evaluation

Detection at the operating point. At its calibrated threshold, the Isolation Forest flags 0.9 % to 1.7 % of frames carrying the six tested faults, close to its 1.2 % nominal-frame flag rate, with an AUROC of 0.54 to 0.77 against nominal frames (Table 4.8); the Mahalanobis and nearest-neighbor scores behave alike. A representation pooled for forecasting is largely insensitive to these image faults. The model-free dark-pixel fraction flags 22.9 % of opaque occlusions and almost nothing else. The per-position frame maximum ranks the faults far better (AUROC 0.84 to 0.97) but flags 1.8 % of nominal 2016 frames after calibration at 1 %, the rim effect described above. The exclusion flag removes that effect. It flags 1.2 % of nominal frames, 85.4 % of banding and 79.2 % of core occlusions, meeting the operating-point criterion of at least half of the banding and occlusion frames at no more than 2 % of clean frames, with lower rates for insects (33.0 %) and soiling (39.8 %). The global median alone detects 79.6 % of banding, while the core maximum responds to compact occlusions. Excluding the rim cells costs sensitivity there, where 35.9 % of rim occlusions and 8.5 % of rim soiling are flagged.

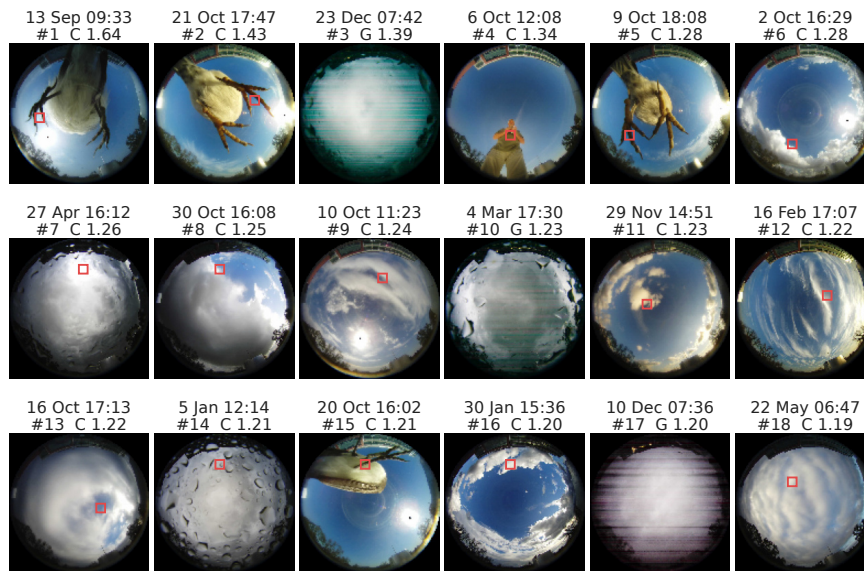


Fig. 4.8. Highest-scoring real Folsom 2016 frames, one per day, ranks 1–18. Titles give date, time, statistic (G, global median; C, core maximum), and score relative to its threshold. Original cell boxes are retained.

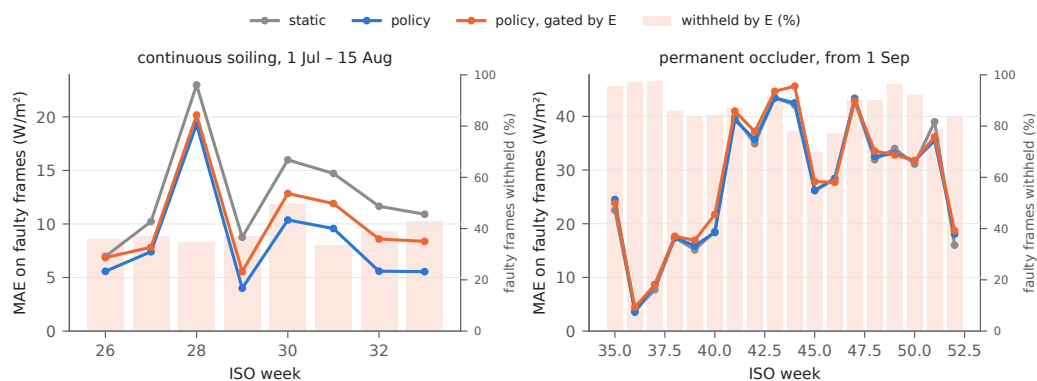


Fig. 4.9. Weekly MAE on the faulty frames under continuous soiling (left, 1 July to 15 August) and under the permanent occluder (right, from 1 September): the static model, the reduced-step policy, and the same policy with the exclusion flag active (E in the legend); bars give the share of faulty frames the flag withholds each week. The ungated policy converges toward its clean-year error on the same dates; the gated policy, denied the frames, does not.

What the flag finds on the real stream. Figure 4.8 shows the 18 real 2016 frames with the highest exclusion scores, one per day. Near the top are apparent sensing-path events, four birds on the dome, a person over the camera, three banding frames, a wet-dome frame, and an internal reflection; the remaining frames show rare sky conditions, isolated cumulus, cirrus streaks, dense overcast, and a red sunrise. The score responds to faults and to unusual nominal imagery alike, which is the reason exclusion has to be judged by what it does to learning.

The policy under faults. Tables 4.9 and 4.10 report the static model, the reduced-step policy, and the reduced-step policy with the flag active on the clean year and on

Tab. 4.9. Predictive outcomes of the static model, the reduced-step policy and the policy with framewise exclusion on the Folsom 2016 replay. MAE is in W/m^2 (lower is better); FS is in % (higher is better). Fault-free means no synthetic fault was applied. Continuous soiling lasts 1 July to 15 August; permanent occlusion begins 1 September. The mixed-episode stream also contains a dense-droplet rendering that is not scored in Table 4.8. Counts and paired contrasts appear in Table 4.10.

Variant	Without synthetic fault		Synthetic fault	
	MAE	FS	MAE	FS
<i>clean</i>				
Static	31.51	15.68	–	–
Policy	31.16	19.23	–	–
Policy + flag	31.13	19.17	–	–
<i>mixed episodes (3.9 % of frames)</i>				
Static	31.70	15.69	39.11	6.14
Policy	31.37	19.31	29.83	14.56
Policy + flag	31.32	19.25	30.11	14.54
<i>continuous soiling (14.6 % of frames)</i>				
Static	35.63	15.69	13.10	7.40
Policy	35.31	19.17	8.62	21.14
Policy + flag	35.19	19.14	10.40	16.55
<i>permanent occluder (29.7 % of frames)</i>				
Static	33.68	16.01	26.90	13.70
Policy	32.97	19.87	27.05	16.23
Policy + flag	32.92	19.82	27.92	14.80

the three corrupted copies of it. Three findings follow, in order of consequence. First, updating on the faulty frames leaves fault-free forecast skill at the clean-year level. On the mixed-episode stream, the policy makes 4,423 updates on faulty frames, 35 % more than the same frames attract when clean, and its forecast skill on fault-free frames is within 0.08 points of the clean-year run; the streams with 1, 5 and 15 % of frames faulty give differences of -0.01 , -0.05 and -0.03 points. The irradiance history remains available on every frame, so what these runs establish is the value of keeping the multimodal observation in the learning stream. Second, exclusion changes prediction little where faults are transient. On the clean year the flag withholds 1.2 % of frames and changes skill by -0.06 points ($[-0.10, -0.00]$); on the mixed stream it withholds 82 % of banding frames and 78 % of occlusions during their episodes, removes a third of the updates made on faulty frames, and changes fault-free skill by -0.06 points, a smaller loss than the 0.15 points ($[-0.26, -0.03]$) that the embedding-distance gate of the originating publication costs on the same replay by withholding frames whose embedding lies beyond a threshold fitted on source data.

Third, exclusion harms adaptation to persistent faults. Under continuous soiling, the flag withholds 38 % of the soiled frames and under the permanent occluder 87 %, and the ungated policy learns through both. Figure 4.9 follows its error on the faulty

Tab. 4.10. Update counts and paired forecast-skill contrasts for the runs in Table 4.9. Withheld is the percentage of synthetic-fault frames excluded, or of all nominal frames for the clean stream. Δ FS is in percentage points on faulty frames, Policy minus Static in Policy rows and Policy + flag minus Policy in flagged rows. Brackets give the 95 % day-block bootstrap intervals.

Variant	Updates total	On faulty	Withheld (%)	Δ FS [95 % CI]
<i>clean</i>				
Static	0	–	–	–
Policy	103435	–	–	–
Policy + flag	100791	–	1.2	–
<i>mixed episodes (3.9 % of frames)</i>				
Static	0	–	–	–
Policy	105116	4423	–	+8.42 [+0.52, +20.94]
Policy + flag	100901	2938	30	-0.01 [-0.29, +0.23]
<i>continuous soiling (14.6 % of frames)</i>				
Static	0	–	–	–
Policy	106829	6460	–	+13.74 [+4.61, +42.12]
Policy + flag	99985	2878	38	-4.59 [-13.61, -1.65]
<i>permanent occluder (29.7 % of frames)</i>				
Static	0	–	–	–
Policy	103971	31143	–	+2.53 [+1.85, +3.68]
Policy + flag	74602	3854	87	-1.43 [-1.93, -1.03]

frames week by week. Under soiling, its excess over the clean-year run on the same dates halves between the first and the last two weeks of the episode, from 1.4 to 0.7 W/m², and under the occluder its error in the last two weeks is within 1 % of the clean-year run. The gated policy, denied the frames it would learn from, does neither; its excess under soiling grows from 2.1 to 3.4, and under the occluder it stays 4 % above the clean-year run. Over the whole episode the flag costs 21 % of MAE on the soiled frames ([+11, +29], about 4.6 forecast skill points) and 3.2 % on the occluded ones ([+1.9, +5.1], 1.4 points). Against the static model the ungated policy is 13.7 skill points better on the soiled frames, with 34 % lower MAE, and 2.5 points better on the occluded ones ([+1.9, +3.7]), of which the gated policy keeps 9.1 and 1.1 points; on the mixed streams it beats the static model on faulty frames by 4.0, 14.5 and 7.5 points at 1, 5 and 15 % of frames faulty, with the gated policy always between the two.

Maintenance alert. None of the three alert designs meets its criterion. The frame-maximum alert signals within two hours in three of four soiling episodes but is raised on 2.8 % of nominal frames against the 1 % target, because the rolling median raises the rate wherever exceedances cluster. Restricting the score to interior cells lowers the rate to 1.6 % while detecting the same three episodes. The 60-minute median of the global score keeps a 3.9 % nominal alert rate and alerts during only 2.9 % of continuous-soiling frames and 5.0 % of occluded frames, since a persistent fault

covers a minority of the 172 cells and leaves the spatial median insensitive. These results point to per-cell persistence as the design to pursue.

4.3.3 Conclusions

Detection and usefulness for learning are different questions and were answered differently. The per-position flag detects banding and core occlusions at a 1.2% nominal flag rate where the pooled embedding score, which reflects what matters for forecasting, detects almost nothing. Applied as an update gate, the same flag costs little on the clean and mixed-episode streams and harms adaptation to the persistent faults, because the withheld frames carried valid labels and were exactly the frames the policy needed to learn the changed appearance. Under both persistent faults the ungated policy moved toward its clean-year error and the gated policy stayed above it (Fig. 4.9). An image fault therefore motivates a maintenance response, and the tested alert designs show that this needs per-cell persistence instead of a spatial median, but it does not by itself justify withholding the frame from supervised learning.

Limitations.

- Every synthetic fault leaves the irradiance label valid. A soiled pyranometer or a clock offset compromises supervision instead, and neither the detection of such label-path faults nor the appropriate update response is evaluated.
- The faults are rendered, at one site and in one year, with one run per configuration and one draw per schedule, and the banding rendering is more saturated than the real frames that motivated it, so its detection rate is optimistic; the real flagged frames are characterized by inspection only.

4.4 Update affordability through accelerator-backed execution

Update affordability depends on where the backbone forward pass executes, and this is the point at which the deployment work of Ch. 3 starts to determine what adaptation costs. Every element of the execution pattern below is a deployment artifact. The backbone must be quantized to a precision the accelerator admits, compiled through a toolchain that accepts its operator set, and partitioned at the host-accelerator interface, and a model that fails any of those checks cannot be adapted by this pattern, whatever its learning algorithm. The pattern is a property of the encoder-head factorization of Eq. 2.2, requiring only a frozen backbone, a trainable head, and a feature-tensor interface between them, so it is evaluated on standard image-classification benchmarks, which permits comparison with established baselines and isolates the system's contribution from task-specific effects. The frozen backbone

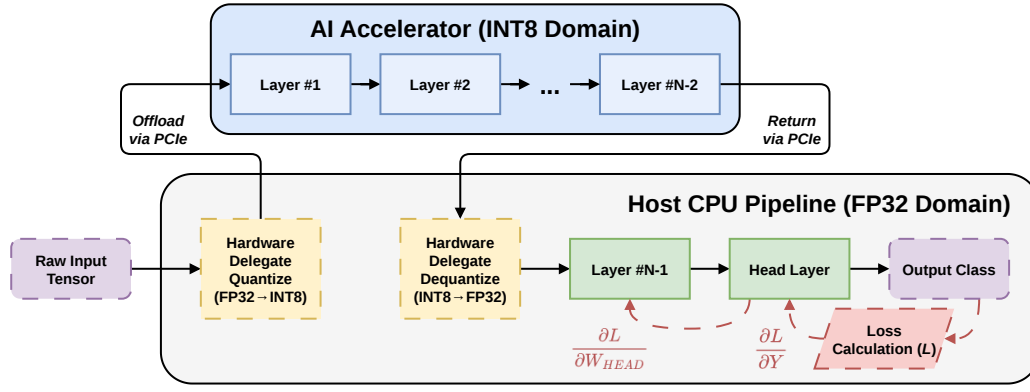


Fig. 4.10. Execution partition of the heterogeneous on-device adaptation pipeline. The frozen backbone executes as an INT8 computation graph on the accelerator, and the resulting feature tensor is transferred to the host, where the FP32 head is adapted.

plays the role of the sky-image encoder $g(\cdot; \theta_b)$ and the trainable head that of the irradiance predictor $h(\cdot; \theta_h)$, and the head updates it accelerates are the most frequent update action of the policy of Sec. 4.2, selected on 34 % to 51 % of buffered samples in Table 4.5 against 9 % to 19 % for encoder updates.

4.4.1 Method

The asymmetry that makes on-device learning tractable is a property of the hardware. Edge accelerators are optimized for the forward pass that dominates an update while lacking the operator support, writable weights, and automatic differentiation a training pipeline needs. They can therefore execute the expensive part of an update and none of the part that makes it an update. At each adaptation step, the accelerator extracts features and streams the feature tensor to the host, which executes the head forward pass, computes the loss, backpropagates through the head only, and applies the optimizer update (Fig. 4.10). The contribution is the Hailo and ORT implementation, in which INT8 preparation with restoration is paired with an FP32 head trained through the on-device training runtime, and features are recomputed at every step, and its characterization across backbones, restoration settings, latency, and energy.

Formal statement and cost model. In the composed form of Eq. 2.2, the heterogeneous configuration replaces the backbone parameters with their quantized counterpart $\tilde{\theta}_b = Q(\theta_b)$, where Q denotes the INT8 quantization and restoration operator specified below, and evaluates that stage on the accelerator,

$$\hat{y} = \underbrace{h(\cdot; \theta_h)}_{\text{host, FP32, trainable}} \left(\underbrace{g(x; \tilde{\theta}_b)}_{\text{accelerator, INT8, frozen}} \right), \quad \Delta \tilde{\theta}_b \equiv 0. \quad (4.5)$$

Tab. 4.11. Frozen-backbone characterization at $3 \times 224 \times 224$ input resolution. MACs and parameter counts describe the portion of the network offloaded to the accelerator; output features is the width of the feature vector delivered across the host interface, and therefore the input dimension of the trainable head. B denotes batch size.

Backbone	MACs (G)	Params (M)	Output features
ResNet18	1.82	11.19	$B \times 512$
EfficientNet-Lite 4	1.35	11.67	$B \times 1280$
MobileNetV3 Large	0.22	4.23	$B \times 1280$
FastViT-SA12	1.38	10.55	$B \times 1024$

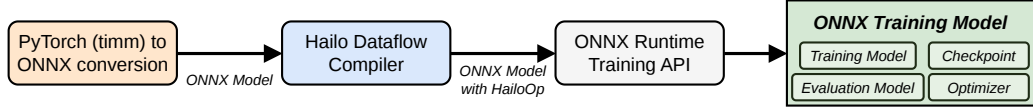


Fig. 4.11. ONNX-based toolchain for heterogeneous adaptation: export, INT8 compilation and restoration, and training-graph generation.

The trainable subspace is $\mathcal{S} = \{0\}^{|\theta_b|} \times \mathbb{R}^{|\theta_h|}$ as in Eq. 4.1, so the chain rule is required only along the head, $\nabla_{\theta_h} \mathcal{L} = (\partial \mathcal{L} / \partial h)(\partial h / \partial \theta_h)$, and the accelerator needs to support inference only. Let C_g^{fwd} and C_h^{fwd} denote the forward cost of backbone and head and C_h^{bwd} the head backward cost. A host-only head update that recomputes features costs $C_{\text{host}} = C_g^{\text{fwd}} + C_h^{\text{fwd}} + C_h^{\text{bwd}}$, and the heterogeneous configuration replaces the first term with its accelerated counterpart $\tilde{C}_g^{\text{fwd}}$, giving a per-update speedup

$$S = \frac{C_g^{\text{fwd}} + C_h^{\text{fwd}} + C_h^{\text{bwd}}}{\tilde{C}_g^{\text{fwd}} + C_h^{\text{fwd}} + C_h^{\text{bwd}}}. \quad (4.6)$$

Two limiting regimes are shown in the total step times below. When the backbone dominates, $S \rightarrow C_g^{\text{fwd}} / \tilde{C}_g^{\text{fwd}}$ and the realized speedup approaches the raw accelerator speedup; when the head terms dominate, $S \rightarrow 1$ however fast the accelerator is. The benefit of the pattern is therefore governed by the ratio of frozen-backbone cost to trainable-head cost, and feature transfer and execution overhead enter only through the measured forward-cost terms.

Backbones. Four ImageNet-pretrained backbones from PyTorch Image Models [134] are evaluated, ResNet18 [27], EfficientNet-Lite 4 [30], MobileNetV3 Large [28], and FastViT-SA12 [31], each with a single linear classification head trained on the emitted feature vector. Table 4.11 gives the two quantities that enter Eq. 4.6, the MACs offloaded from the host and the feature width that sets the input dimension of the head left there. ResNet18 combines the highest operation count with the narrowest feature vector, the backbone-dominated regime, whereas MobileNetV3 Large has about $8\times$ fewer backbone operations and a $2.5\times$ wider vector, leaving relatively more work on the host.

Tab. 4.12. Evaluated performance restoration strategies and their configurations [18]. Eq. denotes channel equalization, IBC iterative bias correction, FT knowledge-distillation fine-tuning, and AR the AdaRound algorithm.

#	Techniques	Cost	Configuration details
0	None	–	–
1	Eq.	low	–
2	Eq. + IBC	low	64 unlabeled images
3	Eq. + FT	medium	4 epochs & 1024 images
4	Eq. + AR	high	320 epochs & 256 images on all layers

Conversion, quantization, and restoration. The subgraph-delegation path of Sec. 2.2.3 places the partition at the encoder–head interface, and the Hailo Dataflow Compiler 3.31.0 [135] prepares the frozen stage, whose output terminates back-propagation (Fig. 4.11). The practical difficulty is feature fidelity, since converting the backbone from FP32 to INT8 can degrade the representations from which the host head learns, so the five restoration configurations of Table 4.12 are applied within the compilation flow with a calibration subset of 1,024 ImageNet validation images [110]. They rise in cost. Channel equalization reduces scale imbalances across adjacent layers [43], iterative bias correction compensates mean shifts in quantized activations from 64 unlabeled images [44], knowledge-distillation fine-tuning trains the quantization degrees of freedom against the floating-point network [46], and AdaRound optimizes the weight-rounding decisions to minimize local task loss [45].

4.4.2 Evaluation

Setup. Adaptation quality is measured on CIFAR-100 [136] and Oxford-IIIT Pet [137], with images resized to 224×224 , normalized with ImageNet statistics, and any augmentation kept identical across devices. All runs use the cross-entropy loss and AdamW at a fixed learning rate of 10^{-3} , and report test accuracy after five epochs at batch size 1 from a single seed. The heterogeneous pipeline uses a Raspberry Pi 5 host [81] connected over M.2 to a Hailo-8L accelerator [83]; the baselines are the Raspberry Pi 5 CPU alone, and the NVIDIA Jetson Orin Nano GPU [84]. All runs use ORT on-device training v1.22.1 [40]; in the heterogeneous configuration, the backbone forward pass executes through the Hailo execution provider (HailoRT v4.23.0) [138], and the head forward pass, loss, backward pass, and optimizer update execute on the host CPU. A training step comprises feature extraction, head forward pass, loss, head backward pass, and optimizer update. Energy per step comes from an external inline meter sampling at 5 Hz on the Raspberry Pi 5, for both the CPU baseline and the accelerator-backed configuration, and from on-module sensors read through `tegrastats` at 1 Hz on the Orin Nano in the MAXN power mode with `jetson_clocks` enabled, without

Tab. 4.13. End-to-end head-only fine-tuning results with batch size 1. *Acc.* is test accuracy after five epochs, *Lat.* is training-step latency (ms/step) and *E* is energy per training step (mJ/step). For the heterogeneous pipeline, Hailo (# · , # ·) gives the restoration strategy of Table 4.12 with the highest test accuracy for each dataset; Fig. 4.12 shows all five. Best value per column in bold.

Model	Hardware	CIFAR-100			Oxford-IIIT Pet		
		Acc. (%)↑	Lat. (ms)↓	E (mJ)↓	Acc. (%)↑	Lat. (ms)↓	E (mJ)↓
ResNet18	RPi 5 (CPU)	64.68	92.85	525.65	86.97	102.84	561.25
	Orin Nano (GPU)	64.67	13.70	128.25	87.00	9.22	128.13
	Hailo (#4, #4) + RPi 5	64.62	6.04	38.65	86.94	7.17	46.35
EfficientNet Lite 4	RPi 5 (CPU)	58.48	178.28	1,042.91	83.37	194.02	1,064.18
	Orin Nano (GPU)	58.80	23.56	234.09	83.43	24.39	184.00
	Hailo (#3, #4) + RPi 5	60.23	23.91	124.64	84.08	23.01	111.99
MobileNetV3 Large	RPi 5 (CPU)	68.26	57.33	286.97	88.88	52.71	315.63
	Orin Nano (GPU)	68.10	9.49	110.04	88.83	9.30	102.19
	Hailo (#4, #4) + RPi 5	55.53	13.73	78.22	87.00	12.31	65.37
FastViT-SA12	RPi 5 (CPU)	71.19	223.89	1,224.60	91.88	229.26	1,228.60
	Orin Nano (GPU)	71.20	21.92	306.73	91.88	27.31	220.37
	Hailo (#3, #3) + RPi 5	50.19	22.81	119.64	83.54	21.84	105.80

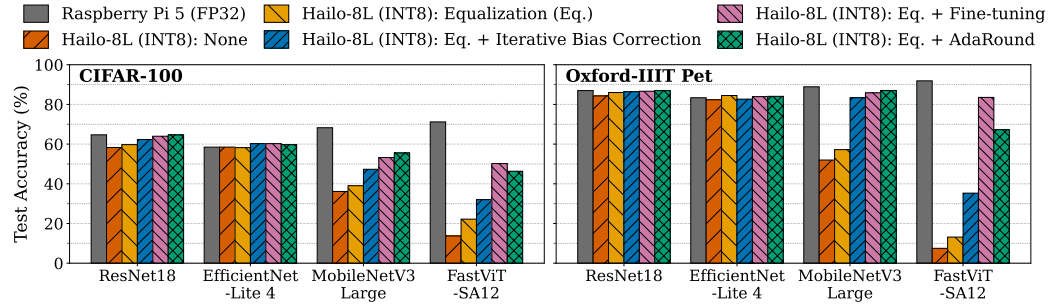


Fig. 4.12. Test accuracy after five epochs of head-only fine-tuning under each of the five INT8 restoration strategies of Table 4.12, across the four backbones and both datasets. Strategy 0 is unrestored post-training quantization. The spread across strategies is widest for the quantization-sensitive backbones.

idle-power subtraction; because module telemetry excludes the carrier-board losses that an inline meter includes, the margin against the Orin Nano indicates a scale, whereas the margin against the CPU baseline is like for like.

Results. Table 4.13 summarizes the trade-off for the restoration strategy with the highest test accuracy per architecture and dataset; Fig. 4.12 shows all five. For ResNet18 and EfficientNet-Lite 4, the accuracy of the heterogeneous configuration stays close to or above the floating-point references while the per-step work falls sharply. On CIFAR-100, ResNet18 goes from 92.85 ms and 525.65 mJ per step on the CPU to 6.04 ms and 38.65 mJ with the accelerator, at 64.62 % against 64.68 % accuracy, and EfficientNet-Lite 4 reaches a latency comparable to the Orin Nano at 53 % to 61 % of its energy. The two quantization-sensitive backbones obtain a comparable speedup at a cost in accuracy. FastViT-SA12, with self-attention, falls from 71.19 % on the CPU baseline to 50.19 % on CIFAR-100, and MobileNetV3 Large, with Hard-Swish activations, from 68.26 % to 55.53 %, even under the best

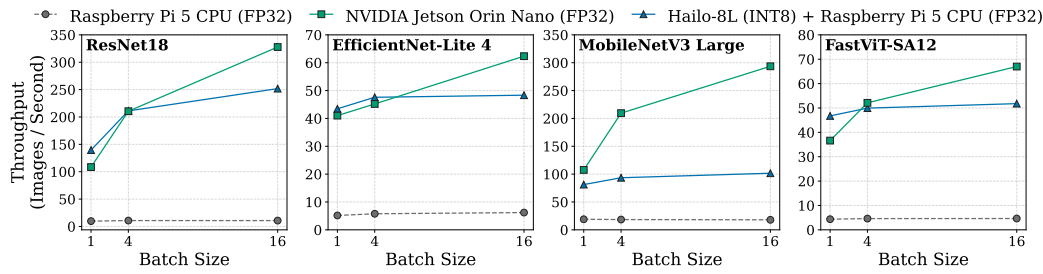


Fig. 4.13. Training throughput against batch size on Oxford-IIIT Pet, in images per second, for the CPU-only baseline, the edge GPU baseline, and the accelerator-backed pipeline. One panel per backbone; higher is better.

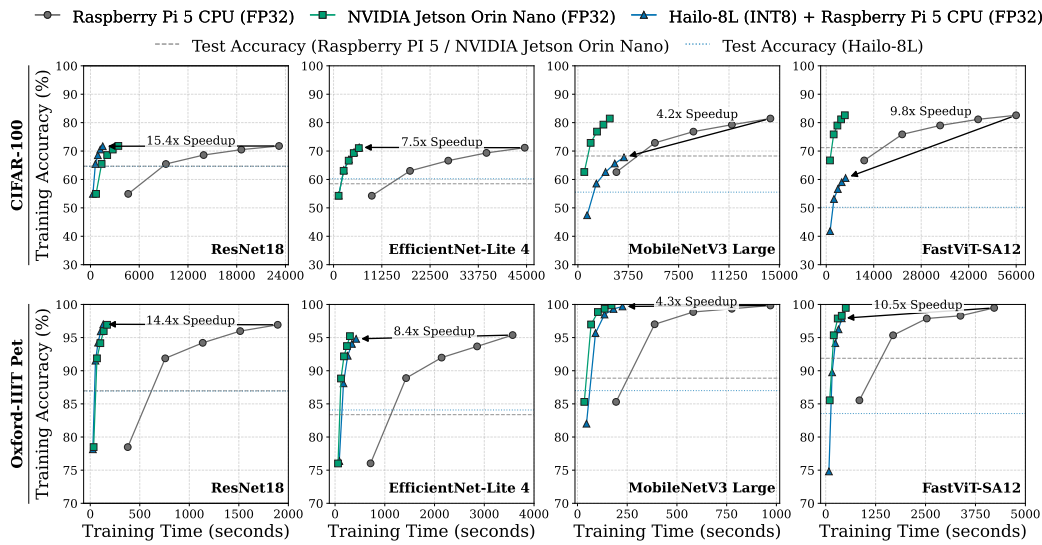


Fig. 4.14. Time-to-accuracy convergence for the four backbones on CIFAR-100 and Oxford-IIIT Pet, one panel per pairing. Curves give training accuracy against wall-clock time under head-only fine-tuning at batch size 1, markers indicate the end of each epoch, and horizontal dashed lines give final test accuracy. The three execution configurations of Table 4.13 are compared.

restoration strategy, so for these two the speedup is obtained on a model that has lost a fifth to a third of its accuracy on CIFAR-100. Since varying the restoration strategy systematically changes the final accuracy (Fig. 4.12), model preparation matters more for adaptation quality in this regime than raw computational capacity, and preserved accuracy under INT8 preparation is the condition for using the measured speedup.

Where the speedup goes. ResNet18 attains up to $15.4\times$ the CPU-only training speed and MobileNetV3 Large about $4\times$, which agrees with the partition model, since a light backbone with a wide feature vector leaves more relative work in the host head. The baseline recomputes features for every training sample. Where fixed features can be cached, or retained with auxiliary inputs during frozen-encoder inference, the head update needs no backbone pass, and the accelerator shortens inference rather than the update. This distinction applies when transferring these

measurements to delayed supervision. Figure 4.13 extends the batch-one results to batches of 4 and 16. The CPU-only baseline improves little, because feature extraction and updating share the host, whereas the heterogeneous path gains through batch size 4 with smaller improvements thereafter. Against the Orin Nano, the batch-one timing advantage is workload-dependent, ResNet18 leads on both datasets, EfficientNet-Lite 4 and FastViT-SA12 lead only on Oxford-IIIT Pet, and MobileNetV3 Large trails on both. The energy results are less equivocal. The accelerator-backed configuration consumes the least energy per training step in all eight backbone-dataset cells, 3.7 to $13.6\times$ below the CPU baseline on the same instrument and 1.4 to $3.3\times$ below the edge GPU, including on MobileNetV3 Large where it is the slower of the two. This is the signature of an inference-centric design, and it suits on-device adaptation, where large batches are undesirable because they inflate buffering and update latency. Figure 4.14 connects the per-step savings to time to accuracy, and the configurations with close reported accuracy reach it in a fraction of the CPU-only time.

4.4.3 Conclusions

Freezing and quantizing the backbone gives high-throughput, energy-efficient feature extraction on the coprocessor while the head keeps a conventional training pipeline on the host. The result is a statement about deployment as much as about adaptation, since the benefit scales with the ratio of frozen-backbone cost to trainable-head cost, a property of the architecture and of the precision and partition chosen at installation, and the pattern is beneficial only where the INT8 backbone keeps sufficient feature fidelity. For ResNet18 it cuts the per-step time by $15.4\times$ and the energy by $13.6\times$ at unchanged accuracy, and it is the most energy-efficient configuration in every tested cell. For the two quantization-sensitive backbones a comparable speedup comes with an accuracy loss that stronger restoration, better calibration or selective higher precision would have to remove. The conversion pipeline, the five restoration configurations and the batch-scaling measurements are reported in full in [18].

Limitations.

- Peak update-time memory is not quantified, so the evaluation reports latency, throughput, and energy only; terminating gradients at the interface avoids retaining backbone activations, but the resulting saving in peak process memory needs measurement.
- The workload is image classification; the restoration strategy in Table 4.13 is the best of five on the test split, and every result is a single seed. The benefit for the forecasting head is argued through the cost model, and the reduced-step encoder action of Eq. 4.4, which backpropagates into the last two encoder stages, lies outside this pattern.

4.5 Summary and the limits of local adaptation

The chapter examined conditional updates under delayed labels, a sensing-fault monitor in front of them, and an accelerator execution path for the head updates, related through the encoder-head interface. Within the evaluated configurations, five lessons are supported.

1. Adaptation under delayed supervision justifies its update budget where relocation has opened a gap to the local statistical reference, which the policy closes at both unseen sites within the idle time of the acquisition period. At the source site, the frozen model is difficult to improve on in absolute error, so there the policy is justified by its safety and low update cost rather than by a large gain.
2. Test the update subspace together with the learning rate. Restricting the trainable encoder stages alone left the source-site deterioration in place; reducing the encoder step on that restricted subspace removed it, and the two thresholds then set an FS-compute trade-off rather than a tuned optimum.
3. Keep a local statistical reference when assessing relocation. A transferred forecaster can lose its advantage over Smart Persistence on absolute error while keeping it on RMSE, and the storage proxy shows the operational form of that difference.
4. Evaluate exclusion by its learning consequences as well as by detection. Withholding the tested persistent image faults impaired recovery while their labels stayed valid, so a detected fault motivates maintenance, not exclusion from supervised learning.
5. Profile update memory as well as latency. For the forecaster, the increase over inference is small for the conditional policy and the delayed-supervision comparators and large only for the two test-time adaptation methods, while the accelerator path's saving in peak memory remains to be measured.

Evidence for RQ2 and RQ3. The reduced-step policy improves on static inference at every evaluated site, is slightly better than sliding-window adaptation on campaign MAE and within a point of it on forecast skill, and does so at lower per-frame energy on both devices and within the idle time of the acquisition period (Sec. 4.2.3). Input admissibility is settled by a concrete result rather than a rule, since the tested persistent corruptions remained useful learning data with valid labels. The accelerator study demonstrates the execution gain of the frozen-encoder partition for the convolutional backbones that keep their accuracy under INT8, which is the pattern the head action of the policy maps onto. Together, the results support the conditional-adaptation and hardware-aware execution parts of the hypothesis for the tested configurations.

The limits of local adaptation. Local adaptation closes part of the gap that relocation opens, and only part. At Poznań the reduced-step policy restores forecast skill to 18.9 %, close to the 19.2 % it holds on the held-out source year, whereas at Karpicko it reaches 12.3 %, and its campaign MAE at both Polish sites remains about twice the Folsom value. The device learns only from its own delayed labels. Hence, a site whose stream is short, seasonally narrow, or partly corrupted has correspondingly little to learn from, and Sec. 4.3 shows that even faulty frames carry information that should not be discarded. The evidence left unused is the experience of other sites that face the same sky regimes with different cameras and seasons. Chapter 5 asks what such peers can add to a model that already adapts to its own observations, and at what communication cost, measured relative to isolated local learning and to parameter exchange. It uses a scheduled daily learner with a trainable encoder and replay rather than this chapter’s per-sample policy. On the same populations, that learner is 3.5 and 2.2 skill points below this chapter’s policy at Folsom and Poznań and level at Karpicko, at a higher daily update cost on the Raspberry Pi 5 and with the lower MAE at the two Polish sites. The collaboration increments of the next chapter are therefore measured relative to a local reference that this policy exceeds on forecast skill but not on every metric.

Communication-limited collaboration across deployed devices

Chapter 4 closed with the limits of local adaptation. A deployed forecaster learns only from its own delayed labels, and the sites that need adaptation most, those farthest from the training distribution, have the least local experience available. This chapter addresses RQ4 in Sec. 1.2 by asking how compact knowledge exchange between sites changes predictive performance and collaboration cost relative to isolated local adaptation and to parameter exchange. The contribution is regime-stratified prototype exchange for continuous forecasting (C5). Physically defined irradiance regimes group the learned all-sky-image representations, aggregated prototypes return to each client as a personalized mixture, and the mixture guides local regression through centroid alignment. Three findings follow. The exchanged object is far smaller than a model state, forecast skill is preserved relative to isolated local adaptation where parameter exchange loses it, and an own-prototype control separates regularization from peer information. As in Sec. 3.4, the forecasts serve photovoltaic and storage decisions, and the chapter considers their predictive and communication outcomes.

5.1 Collaboration protocol and evaluation setup

5.1.1 What a single device cannot learn

Centralizing the raw streams would let one site's experience reach the others, but all-sky imagery is high-bandwidth and continuous, and governance constraints can preclude export. FL addresses this by exchanging model updates or compact summaries under server coordination while measurements stay local (Sec. 2.5.1) [61]. The question examined here is whether the predictive gain of such exchange justifies its cost when the participating sites differ systematically, and the answer has the three parts that the collaboration protocol of Sec. 2.6.1 names. Predictive preservation is judged against isolated local adaptation at each site, primarily on FS with MAE and RMSE alongside; communication efficiency against full-state parameter exchange; and participation resilience under controlled client dropout. The scope is server-coordinated collaboration, in which a coordinator aggregates uploaded objects and returns an aggregate; fully peer-to-peer schemes, fairness, and incentive mechanisms lie outside it. All evaluated methods keep raw samples local. A

prototype or a parameter update is still a function of local data. It can expose information [70], and secure aggregation [71] or differential privacy [72] would add formal protection at additional cost (Sec. 2.6.1).

5.1.2 Evaluation setup

Three monitoring sites act as clients, Folsom on its held-out 2016 stream, and the Poznań and Karpicko campaigns of Table 2.4; the ResNet50 checkpoint of Ch. 4 is the source model at every site (Table 2.5). Folsom is the source domain, since the checkpoint was trained on its 2014 and 2015 partition, whereas the Polish sites were never seen in training; the difference is visible in Folsom’s much lower absolute errors, and it means that transfer from Folsom and transfer to Folsom are not symmetric operations. The campaign is a season-aligned replay over a normalized calendar from 1 January to 31 December, formed by omitting the year from the timestamps so that the three streams advance together through the seasons; the 29 February of Folsom’s 2016 is not replayed. Coverage is uneven. Folsom uploads a prototype set in all 52 scheduled rounds, Karpicko in 46, and Poznań in 33, with none between 1 July and 18 October.

MAE, RMSE, and FS against Smart Persistence are reported per site over the regime-labeled daylight populations of Table 4.3. Pooling follows Sec. 2.7.6; for daily counts n_d on a common mask,

$$\text{RMSE}_{\text{pooled}} = \sqrt{\frac{\sum_d n_d \text{RMSE}_d^2}{\sum_d n_d}}. \quad (5.1)$$

Replication and uncertainty. The five parameter-space configurations use three seeds each, the proposed configuration and the isolated reference five, and Static one checkpoint; ablation, control, and dropout variants state their seed counts with their results. Daily errors are averaged over seeds, and a moving-block bootstrap over days with blocks of seven days and 2,000 date-paired resamples gives 95 % intervals for every contrast, each comparison seeding its own generator. These intervals describe temporal variability over the replay conditional on the fitted runs, and they are not adjusted for the number of contrasts a figure shows. A difference is called supported where its interval excludes zero. Seed-level separation is reported alongside as an exact permutation test on per-seed forecast skill; with five runs against five, complete separation gives $p = 1/252$, the smallest value the design can attain, and with five against three it gives $p = 1/56$. Communication payload and per-client device cost are reported in Tables 5.4 and 5.5; transport and the coordinator are not timed on the devices.

5.2 Regime-stratified prototype exchange for multi-site forecasting

The workload is a structured stress case for collaboration. Sites differ in latitude, climate, cloud motion, and optics, and partly cloudy periods produce different error patterns at different sites, so averaging models can interfere with local specialization. The mechanism below therefore exchanges neither parameters nor raw embeddings but per-regime summaries of the representation, and it is assessed against a strong local-adaptation reference.

5.2.1 Method

Forecasting model and local training procedure. The task is single-step 15-minute-ahead global horizontal irradiance regression with the encoder–head composition of Sec. 4.1.2. Each client performs one local epoch per day over the day’s samples, replays a first-in-first-out buffer of its own 1,000 most recent samples at a ratio of 0.5 to the day’s count, and synchronizes every 7 days, which gives 52 scheduled rounds without an isolated warm-up. The optimizer, learning rate, weight decay, clipping, and batch size follow the client-settings row of Table 2.5, and a fresh optimizer is created for each day’s epoch so that no optimizer state survives an exchange. The encoder is trained locally, including through the alignment term introduced below, and prototypes are extracted after the day’s update, so they describe the model that produced the day’s adaptation. A client without samples in a round still calls the server with an empty prototype set of 1,261 bytes and receives an empty return, so no stale prototype is reused, and it resumes with fresh prototypes on its first day with data. The local rule is the nightly epoch rather than the per-sample conditional policy of Ch. 4, for three reasons. A synchronous round requires all clients to reach the exchange point after a comparable amount of local training, the replay buffer is drawn once per day, and a per-sample rule would let client states diverge between rounds by amounts set by each site’s error distribution. The cost of this choice relative to the stronger local learner is stated with the results.

Regimes and prototype construction. Following the prototype-exchange formulation of [68], each client i accumulates per-class feature sums and counts over the samples of its extraction pass, from which the mean prototype of sky class c is formed,

$$p_i^c = \frac{1}{N_i^c} \sum_{x \in \mathcal{D}_i^c} f_i(x), \quad (5.2)$$

where $f_i(x) = g(x; \theta_{b,i}) \in \mathbb{R}^d$ is the encoder embedding of sample x at client i . $\mathcal{D}_i^c$ is the set of client- i samples assigned to class c , and $N_i^c = |\mathcal{D}_i^c|$.

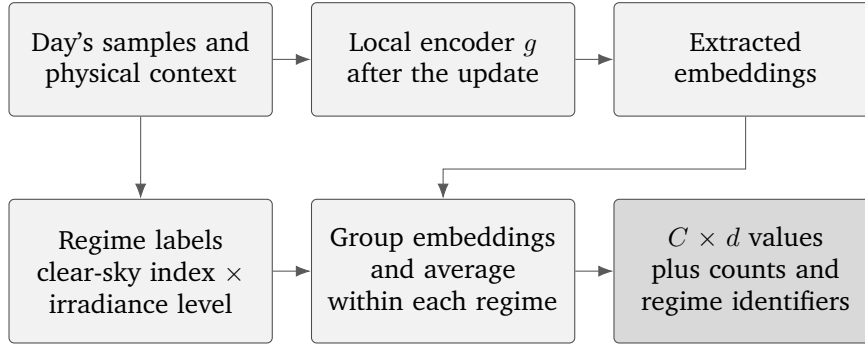


Fig. 5.1. Client-side prototype construction for $C = 12$ regimes. Embeddings are produced by a forward pass over the day's samples after the local update, then stratified by clear-sky index crossed with clear-sky irradiance level (a proxy for solar elevation) and averaged within each regime. The serialized object also carries support counts and class identifiers, which Table 5.4 measures.

Prototypes are computed per regime rather than per client, which restricts alignment to comparable operating conditions, and the regimes are defined physically so that a class index denotes the same sky state at each site. One labeling function assigns each sample its regime from its measured target irradiance and the clear-sky irradiance. The same function serves prototype extraction and the alignment term, so the two cannot drift apart. Samples are stratified by the Ineichen–Perez clear-sky index $k^{cs} = \text{GHI}/\text{GHI}_{cs}$ already used for Smart Persistence (Sec. 2.7.4), with cut points 0.30 and 0.85. Dividing by the clear-sky irradiance of the site and time removes the latitude dependence that a fixed $1,000 \text{ W/m}^2$ denominator would introduce, under which no Polish winter sample could be labeled clear.

The sky class is crossed with four clear-sky irradiance levels. These are cut at 0.15, 0.35 and 0.60 kW/m^2 . The levels serve as a monotone proxy for solar elevation and give $C = 12$. Class balance differs by site. Clear skies form 71 % of Folsom's scored samples, but 23 % of Karpicko's and 32 % of Poznań's. Overcast skies dominate at the Polish sites with 48 % and 38 %.

A sample whose clear-sky irradiance falls below 1 W/m^2 carries no regime and is excluded from prototype construction, from the alignment term, and from the metrics. The clear-sky index is undefined at night, and clamping its denominator would assign night frames to the clear class. The Polish cameras record continuously, and without this exclusion 39.9 % of Karpicko's and 41.0 % of Poznań's clear-class samples would be night frames. Figure 5.1 shows the construction end to end.

Aggregation. Rather than returning a single global prototype, the server returns to each client a personalized mixture of the participating clients' prototypes of the same

class, including its own. With cosine similarity $\text{sim}(\cdot, \cdot)$ over the clients $\mathcal{P}^c$ present for class c ,

$$w_{ij}^c = \text{softmax}_{j \in \mathcal{P}^c} \left(\frac{\text{sim}(p_i^c, p_j^c)}{\sigma^c} + \zeta(\log(1 + N_j^c)) \right), \quad \hat{p}_i^c = \sum_{j \in \mathcal{P}^c} w_{ij}^c p_j^c, \quad (5.3)$$

where σ^c is the standard deviation of the pairwise similarity matrix for class c and $\zeta(\cdot)$ standardizes to zero mean and unit variance. Similarities are computed between the ℓ_2 -normalized prototypes, the mixture is formed from the unnormalized ones, the similarity scale adapts to the spread of the participating prototypes instead of being a fixed temperature, and the support-count term favors better-supported prototypes and vanishes when all counts are equal. In a round with a single uploader, the client receives its own prototype back unchanged, so such a round introduces no peer information.

Alignment objective. The returned mixture changes the client's model through an auxiliary alignment term added to the local objective. Writing $\mathcal{L}_{\text{fc}}$ for the local forecasting loss, $\bar{z}^c$ for the ℓ_2 -normalized mean embedding of the samples of class c in the current batch and $\hat{u}^c$ for the ℓ_2 -normalized returned prototype of class c , the client minimizes

$$\mathcal{L}_i = \mathcal{L}_{\text{fc}} + \lambda \mathcal{L}_{\text{fc}}^{\text{sg}} \mathcal{L}_{\text{align}}, \quad \mathcal{L}_{\text{align}} = -\frac{1}{|\mathcal{C}_{\mathcal{B}}|} \sum_{c \in \mathcal{C}_{\mathcal{B}}} \log \frac{\exp(\bar{z}^{c\top} \hat{u}^c / \tau_a)}{\sum_{c' \in \mathcal{C}} \exp(\bar{z}^{c\top} \hat{u}^{c'} / \tau_a)}, \quad (5.4)$$

where $\mathcal{C}_{\mathcal{B}}$ are the classes present in the batch, $\mathcal{C}$ the classes for which a prototype was received, $\tau_a = 0.5$, $\lambda = 1.0$, and $\mathcal{L}_{\text{fc}}^{\text{sg}}$ the forecasting loss with its gradient stopped, which scales the alignment term to the current task-loss magnitude. The returned prototype is held fixed between synchronization rounds and detached from the graph, so the gradient flows through $\bar{z}^c$ alone. Two choices in this objective are specific to a continuous target. Alignment acts on batch centroids rather than on individual samples, so that the within-regime variation a regression head needs is retained, and it takes a contrastive form, so that the matched regime is pulled toward its prototype. In contrast, the other received regimes act as negatives. Collapse of the representation is not a maximum of this loss, since if every normalized returned prototype coincided, the logits would be equal, the loss would be $\log |\mathcal{C}|$ and its gradient zero. The per-sample squared-distance alignment familiar from classification behaves differently on this task, as Sec. 5.2.2 shows. Reusing the batch embeddings keeps alignment inside the local epoch, whereas post-update prototype extraction needs a separate pass over the day's samples, and both are charged in the device profile.

Baselines and exchanged objects. Five established federated strategies are compared, chosen so that each of the four response families of Sec. 2.5.2 is represented (Table 5.1). FedAvg averages parameters by local sample count [61]; FedProx adds a

Tab. 5.1. Collaborative-learning design space and exchanged state. Values per direction are the conceptual count, excluding framing; Table 5.4 reports measured serialized round trips. $|\theta|$ denotes full model-state size, $\theta_{\setminus \text{BN}}$ the state excluding retained batch-normalization quantities, $d = 2048$ the embedding width, and $C = 12$ the number of regimes. The final column gives the saving that a payload-aware implementation of each mechanism would achieve against full-state exchange, measured by masking the retained keys before serialization. The runs evaluated here upload the full state for all five parameter-space methods, so the column is an upper limit of the saving. All methods keep raw samples local.

Mechanism	Exchanged object	Values per direction	State kept local	Payload saving
FedAvg / FedProx	Full model state	$ \theta $	No personalized parameter subset specified	0 %
FedEMA	Smoothed model state	$ \theta $	Half of the local state, through interpolation ($\alpha = 0.5$)	0 %
FedPer	Shared encoder state	$ \theta_{\text{shared}} $	Task head	2.2 %
FedBN	Non-BN model state	$ \theta_{\setminus \text{BN}} $	BN parameters and statistics	0.5 %
Proposed	Regime prototypes	$d \cdot C$	Encoder and task head	99.9 %

proximal penalty [64]; FedBN keeps normalization parameters and running statistics local [65]; FedPer keeps the task head local [66]; and FedEMA blends the returned aggregate into each client’s state with equal weight [67]. Static, with no on-site adaptation, and Isolated, with per-site continual adaptation and no exchange, anchor the comparison; Isolated is the predictive-preservation and operational-value reference (Sec. 2.7.6), and full-state exchange is the communication reference. All methods share the local training procedure above, aggregation is weighted by each client’s samples in the round, FedProx uses $\mu = 0.01$ and FedEMA $\alpha = 0.5$, and each baseline runs at this single setting without tuning.

Communication-cost model. Feasibility of collaboration depends on synchronization overhead as much as on predictive gain, so the task metrics are complemented by a cost model with four quantities, the uplink and downlink payload per synchronization round $S_{\text{up}}, S_{\text{down}}$, the client-side overhead T_{client} of preparing and consuming the exchanged object, the coordinator aggregation time T_{server} , and the end-to-end round time under a fixed reference network profile,

$$T_{\text{round}} = \max_{i \in \mathcal{K}} \left(T_{\text{client},i} + \frac{8S_{\text{up},i}}{B_{\text{up},i}} \right) + T_{\text{server}} + \max_{i \in \mathcal{K}} \left(\frac{8S_{\text{down},i}}{B_{\text{down},i}} \right), \quad (5.5)$$

where $\mathcal{K}$ is the set of participating clients and B_{up} , B_{down} the available bandwidths in bits per second, with S in bytes and T in seconds. The model omits overlap, queueing, and retransmission. Payloads are the lengths of the serialized objects recorded as they crossed the device interface, so they include counts, identifiers, and framing.

5.2.2 Evaluation

Local adaptation as the reference. Isolated local adaptation lowers MAE relative to the static checkpoint by 24.1 % at Karpicko and 19.6 % at Poznań and raises forecast skill from 1.63 to 12.32 and from 10.73 to 16.74, whereas at Folsom, the source site, it is 0.08 points better on forecast skill and 0.9 % worse on MAE (Table 5.2). The nightly epoch with replay and the per-sample policy of Ch. 4 are two different local rules on the same checkpoint and population, and neither dominates. The per-sample policy has the higher skill at Folsom and Poznań (19.23 and 18.91 versus 15.76 and 16.74) at about 70 s of daily update compute on the Raspberry Pi 5, compared with about 420 s for the nightly epoch (Tables 4.5 and 5.5). The nightly epoch has the lower MAE at the Polish sites, 55.94 and 60.99 versus 58.58 and 65.90 W/m². The per-sample rule reduces the large errors that forecast skill weighs, and the nightly epoch reduces the typical ones that MAE weighs. The nightly epoch is the rule compatible with synchronous rounds, and the increments that exchange adds below are smaller than the skill gap between the two local rules at Folsom and Poznań, so collaboration is judged against a well-adapted local model rather than against the static checkpoint.

Predictive preservation. None of the five parameter-space configurations reaches the forecast skill of Isolated at any site (Table 5.2 and Fig. 5.2). The fifteen reductions span 0.59 to 2.42 points, fourteen of the fifteen intervals exclude zero, no method ordering is stable across sites, and keeping the task head or the normalization state local does not change the outcome. The proposed configuration is the only method above Isolated, by 0.294, 0.426 and 0.275 points at Folsom, Karpicko and Poznań. The two target-site intervals exclude zero, Folsom's spans $[-0.072, +0.621]$, and at every site each of the five proposed runs exceeds each of the five isolated runs, the complete separation that gives $p = 1/252$. The secondary metrics show where the gain comes from. MAE is 1.4 % worse than Isolated at Folsom, 0.4 % worse at Poznań, and 0.2 % better at Karpicko. At the same time, RMSE is lower at every site, and decomposed by sky class, the RMSE gain lies entirely in the clear regime, -1.1 , -4.6 and -1.7 W/m² at Folsom, Karpicko and Poznań. In contrast, overcast RMSE is higher by 2.0, 2.7 and 1.0 W/m², while the transient regime is within 0.6 W/m² of Isolated everywhere. The exchange therefore improves the squared-error metrics that forecast skill weighs, and it does so in the clear regime, in which the source site has the largest share of samples (71 % against 23 % and 32 %, Sec. 5.2.1).

Tab. 5.2. Performance comparison of collaborative knowledge-sharing methods for 15-minute-ahead solar irradiance forecasting, per site, over a normalized calendar year. MAE and RMSE in W/m², lower is better; FS against the Smart Persistence row of the same site, higher is better. n is the number of seeds averaged. Isolated, the adaptive local-training baseline without collaboration, is the predictive-preservation reference and is set in italics. Best value per column in bold. Smart Persistence is scored on the same population as the models and is the forecast-skill denominator, so its forecast skill is zero by construction; it is a reference row excluded from the bolding. The Static row is the checkpoint of Table 4.3.

Site	Method	n	MAE ↓	RMSE ↓	FS (%) ↑
Folsom	Smart Persistence	–	35.95	86.10	0.00
	Static	1	31.51	72.60	15.68
	<i>Isolated</i>	5	<i>31.78</i>	<i>72.53</i>	<i>15.76</i>
	FedAvg	3	33.24	73.91	14.15
	FedProx	3	33.24	73.95	14.10
	FedBN	3	33.14	74.03	14.01
	FedPer	3	32.62	73.24	14.93
	FedEMA	3	32.76	73.56	14.56
	Proposed (prototypes)	5	32.24	72.28	16.05
Karpicko	Smart Persistence	–	67.19	142.46	0.00
	Static	1	80.40	140.14	1.63
	<i>Isolated</i>	5	<i>60.99</i>	<i>124.92</i>	<i>12.32</i>
	FedAvg	3	63.65	128.21	10.01
	FedProx	3	63.67	128.23	9.99
	FedBN	3	62.64	127.25	10.68
	FedPer	3	63.64	128.36	9.90
	FedEMA	3	62.49	126.74	11.03
	Proposed (prototypes)	5	60.88	124.31	12.74
Poznań	Smart Persistence	–	67.73	138.99	0.00
	Static	1	69.56	124.08	10.73
	<i>Isolated</i>	5	55.94	<i>115.71</i>	<i>16.74</i>
	FedAvg	3	57.97	117.17	15.69
	FedProx	3	58.00	117.24	15.65
	FedBN	3	57.60	116.83	15.94
	FedPer	3	57.81	117.28	15.62
	FedEMA	3	57.36	116.55	16.15
	Proposed (prototypes)	5	56.15	115.33	17.02

What the design provides. Figure 5.3 evaluates the alignment weight on a grid of discrete values from 0.1 to 3.0. The gain over Isolated is highest at $\lambda = 1.0$ at Folsom and Karpicko and lower at the larger tested values, while Poznań still rises within the grid, and $\lambda = 1.0$ is the weight used throughout. Table 5.3 then varies one component of the proposed configuration at a time relative to the same isolated reference, and adds two diagnostics of the representation itself. $\overline{\cos}$ is the cosine

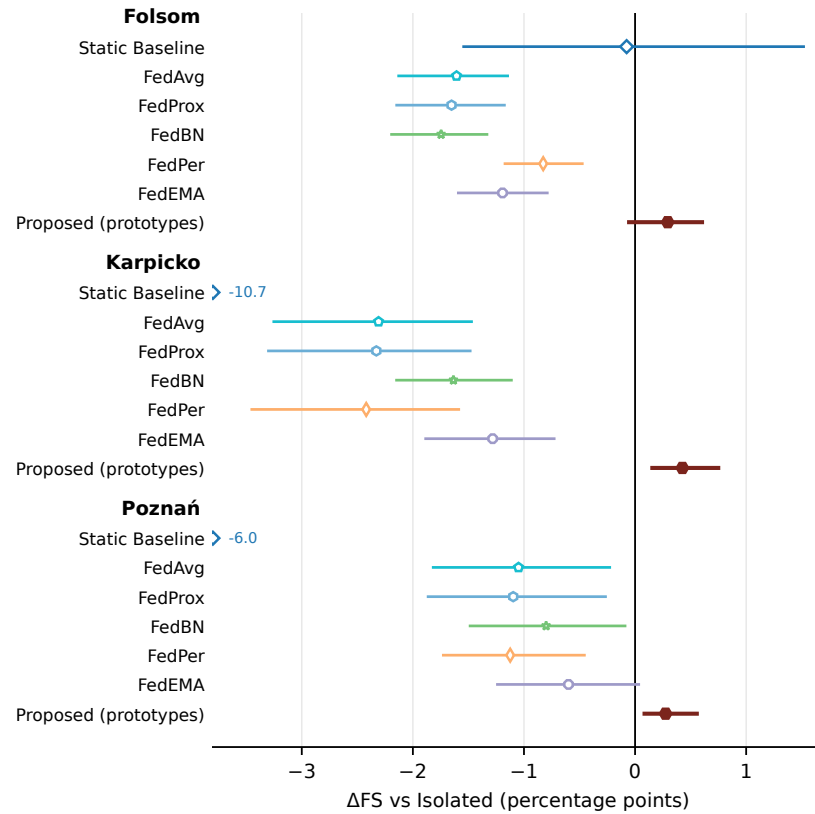


Fig. 5.2. Change in forecast skill relative to isolated local adaptation, per site; positive favors the method. Intervals are the 95 % day-block bootstrap intervals of Sec. 5.1.2 on seed-averaged daily errors (seed counts as in Table 5.2). Static at Karpicko and Poznań lies beyond the axis and is printed. Intervals are unadjusted for the 21 contrasts shown.

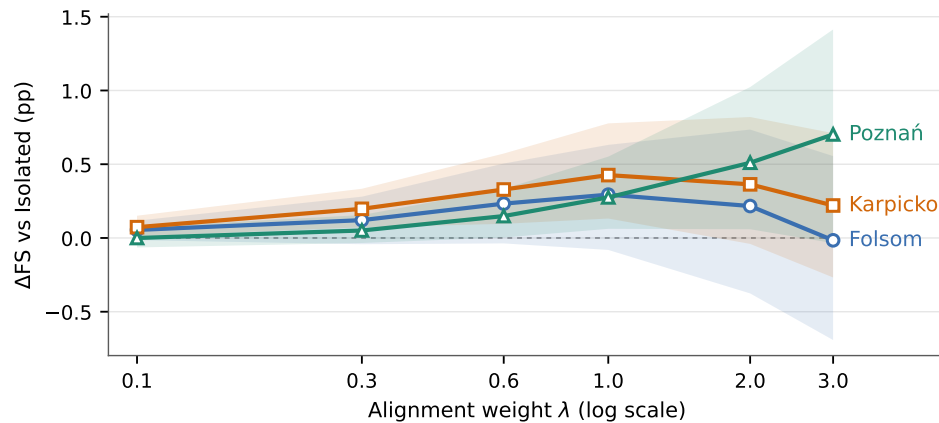


Fig. 5.3. Response of forecast skill to the alignment weight λ , at $C = 12$, against isolated local adaptation. The horizontal axis is the parameter grid of tested values; markers are the evaluated points and the lines only connect them. Shaded bands are 95 % moving-block bootstrap intervals; three seeds per point except $\lambda = 1.0$, which has five.

between pairs of individual sample embeddings, computed during the extraction pass over the regime-labeled samples of the day and averaged over all pairs, then

Tab. 5.3. Component ablation. Variants of the proposed configuration ($C = 12$, centroid contrastive alignment, $\lambda = 1.0$, adaptive attention), each changing one component. ΔFS is the change from isolated local adaptation in percentage points, against the same five-seed reference as Table 5.2. $\overline{\cos}$ and $\|z\|$ are the mean pairwise cosine and the mean norm of the sample embeddings of the extraction pass, averaged over rounds, clients, and seeds as defined in the text. n gives each variant’s seed count. The last row replaces the returned mixture with the client’s own prototype, so that variant exchanges nothing.

Configuration	n	Folsom	Karpicko	Poznań	$\overline{\cos}$	$\ z\ $
Proposed ($C = 12$)	5	0.294	0.426	0.275	0.633	1.746
<i>Alignment objective</i>						
→ per-sample squared distance	3	−18.882	−10.032	−11.611	0.988	1.017
<i>Regime resolution</i>						
$C: 12 \rightarrow 6$	3	0.032	0.287	0.386	0.671	2.048
$C: 12 \rightarrow 3$	2	−0.101	0.113	0.037	0.700	2.201
$C = 1$ (no contrastive signal)	3	0.016	−0.018	0.023	0.778	1.294
<i>Aggregation weighting</i>						
attention → uniform	3	0.330	0.372	0.285	0.651	1.706
attention → fixed $\tau = 0.5$	3	0.297	0.385	0.341	0.636	1.743
<i>Alignment weight</i>						
$\lambda: 1.0 \rightarrow 0.1$	3	0.054	0.073	0.000	0.738	1.401
<i>Alignment target</i>						
peers’ mixture → own prototype (zero payload)	3	0.293	0.114	0.191	0.630	1.720

over rounds, clients, and seeds; it is neither a distance between regime prototypes nor between sites, and a value near one means that the embeddings of different samples point in almost the same direction. $\|z\|$ is the mean norm of the same unnormalized embeddings. The per-sample squared-distance alignment, the form familiar from classification, is the one variant that fails. Over three full-year runs, it lowers the mean embedding norm from 1.746 to 1.017. It raises the mean pairwise cosine from 0.633 to 0.988, reaching 1.000 by the final round, and forecast skill falls by 10 to 19 points, to −3.13 at Folsom, below Smart Persistence; matching representations sample by sample removes the within-regime variation that the regression head needs, whereas the centroid-contrastive objective keeps the cosine between 0.738 at $\lambda = 0.1$ and 0.538 at $\lambda = 3.0$, spreading the representation further as the alignment weight rises. Regime resolution matters less. The coarser partitions cross the three sky classes with two clear-sky irradiance levels cut at 0.15 kW/m² ($C = 6$), use the sky classes alone ($C = 3$), or merge all samples into a single regime ($C = 1$). The site-averaged margin over Isolated falls from 0.332 at $C = 12$ to 0.007 at $C = 1$, where a single received class makes the contrastive term of Eq. 5.4 identically zero, so that row measures the local procedure without alignment; $C = 6$ is within 0.1 points of $C = 12$ on average and above it at Poznań, so the finer partition is preferred for its physical interpretability, not for a resolved gain. The aggregation weights matter least. Uniform weighting and a fixed temperature of 0.5 lie within 0.07 points of the adaptive attention of Eq. 5.3 at every site, with the same cosine and norm diagnostics.

Tab. 5.4. Per-client uplink plus downlink per synchronization round in decimal megabytes, from the recorded lengths of the serialized objects, without compression or network-protocol overhead. The prototype object grows with the number of populated regimes, so its maximum and median over client-rounds are both given. Values per direction count state elements or embedding values, each stored in 32 bits; the serialized objects add support counts, regime identifiers, and framing, about 7 kB for the prototype object and 0.2 MB for the full state. Reduction is the state-to-embedding value-count ratio per direction. Isolated and the own-prototype control of Table 5.3 exchange nothing.

Mechanism	Round trip, maximum (MB)	Round trip, median (MB)	Values per direction	Reduction ($\times$)
Isolated (no exchange)	0.000	0.000	0	–
FedAvg / FedProx / FedEMA (full state)	192.964	192.964	24,091,127	1
Proposed, $C = 12$, $d = 2048$	0.204	0.153	24,576	980

Does the exchange carry information, or only regularize? A client may benefit from consistency to its own earlier representation as much as from its peers, so the last row of Table 5.3 replaces the returned mixture $\hat{p}_i^c$ in Eq. 5.4 with the client’s own prototype p_i^c . This ablation keeps the refresh cadence, class availability, extraction, and gradient detachment of the proposed configuration, sends and receives nothing, and records zero payload, so its contrast with peer exchange is the incremental value of peers, called the peer increment below. The answer differs by site. At Folsom, the source domain, own and peer targets give the same mean gain over Isolated (0.293 versus 0.294 points), so the improvement there is self-regularization. At Karpicko peers add 0.312 points, with a day-block interval of $[+0.135, +0.528]$ and complete separation across seeds ($p = 1/56$); at Poznań they add 0.084 points, with an interval of $[-0.027, +0.204]$. Both configurations keep a mean pairwise embedding cosine near 0.63, far from the concentration of the squared-distance variant, so peer information is not what keeps the representation spread. The control changes how communication efficiency is interpreted. Where a local prototype gives the same outcome, a small payload is not a reason to communicate, and the zero-payload configuration is the configuration to prefer; where peers add a resolved peer increment, as at Karpicko, the payload and the extraction pass measured below are the cost of that increment.

Communication cost. At fixed embedding width d the prototype payload scales with $C \cdot d$ and not with model size (Table 5.4). A full-state round trip carries $2 \times 24,091,127$ values in 192.964 MB, whereas the prototype round trip varies with the number of populated regimes. Its maximum over client-rounds, reached when all twelve regimes are populated, carries $2 \times 24,576$ values in 0.204 MB, $980 \times$ fewer values than the full state, and a serialized payload $946 \times$ smaller. The byte ratio is below the value-count ratio because the prototype object also carries counts, identifiers, and framing (Table 5.4). The median prototype payload over client-rounds is 0.153 MB, $1,260 \times$ below the full state, since a short winter day fills fewer regimes. Transport overhead is not counted; it adds per-message costs that

Tab. 5.5. Per-client cost of one replay day for the three arms on the two target devices, median over seven Folsom days (461 to 850 samples, median 719) after a discarded warm-up day. Phases in seconds are prediction over the day, the local epoch with replay (with the alignment term for the prototype method), post-update prototype extraction, and the client side of the exchange (serialization, deserialization, and loading of the returned object). Energy is the median over days of the daily sum over phases, net of the idle power measured before the run, 1.69 W from the Raspberry Pi 5 PMIC and 3.39 W from Orin tegrastats in the MAXN power mode with `jetson_clocks` enabled; the prototype exchange is too short to sample and adds no energy term. Differences and phase-level energies quoted in the text come from the same per-day logs as medians of paired differences, so they need not equal differences of the table medians. RSS is the process high-water mark in MB, and upload is the serialized object per round, median over the profiled days.

Method	Predict (s)	Train (s)	Extract (s)	Exchange (s)	Energy (J/d)	Peak RSS (MB)	Upload (MB)
<i>Raspberry Pi 5, CPU, four threads</i>							
Isolated	67.7	416.5	–	–	1224	3222	0
FedAvg	66.9	396.5	–	0.33	1228	3194	96.5
Proposed	67.1	414.3	51.3	0.07	1459	3242	0.09
<i>Jetson Orin Nano, GPU</i>							
Isolated	19.5	15.1	–	–	201	4010	0
FedAvg	16.4	15.1	–	0.38	207	4194	96.5
Proposed	16.4	15.9	2.1	0.03	240	4209	0.09

leave these ratios essentially unchanged, and the link-rate threshold below uses the serialized payload as the transferred quantity. The ratios quoted in this chapter use the maximum and are therefore the conservative ones; with the $C = 3$ partition of the ablation, the maximum round trip falls to 0.053 MB. Table 5.5 gives the device work behind this saving, from a matched profile of an isolated, a FedAvg, and a prototype client on the two target devices, each replaying the same seven Folsom days after a discarded warm-up day with the local training procedure of Sec. 5.2.1. The returned objects are produced on the device, so transport and the coordinator are outside the measurement. Training with the alignment term costs 0.8 s more per day than isolated training on the Orin Nano on each of the seven days, and on the Raspberry Pi 5 a median of 17 s more, a paired difference that lies inside the day-to-day variation between the FedAvg and isolated arms on the same days (–82 s to 34 s). The training medians of Table 5.5 (414.3 s and 416.5 s on the Pi) point the other way because they are medians of each method taken separately, not of the paired difference. The prototype-specific work is the extraction pass, 51 s and 151 J per day on the Pi and 2.1 s and 30 J on the Orin, plus a client-side exchange of 0.03 s to 0.07 s against 0.33 s to 0.38 s for the full-state client, and peak process memory is within 5 % across the three arms on each device. The own-prototype control runs the prototype client without the exchange, so the incremental client cost of peers over own prototypes reduces to that exchange plus the untimed transport, with extraction paid either way.

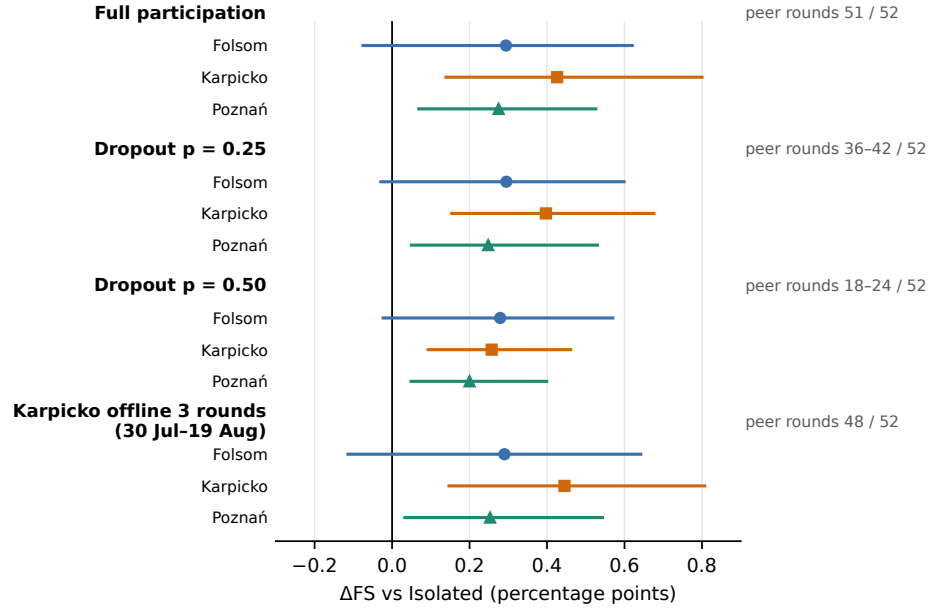


Fig. 5.4. Change in forecast skill relative to isolated local adaptation under controlled client dropout, at the operating point of Table 5.2, with 95 % day-block bootstrap intervals. Dropout probability is a client's independent absence from a given round; the outage withholds Karpicko continuously for 21 replay days, spanning three consecutive rounds. Peer rounds is the range over seeds of rounds, out of 52 scheduled, in which some client received a prototype from a different client.

The cost model links this local work to the network saving. For equal clients and a symmetric effective link rate B , the two round times of Eq. 5.5 are equal at

$$B^* = \frac{8\Delta S}{\Delta T_{\text{client}} + \Delta T_{\text{server}}}, \quad (5.6)$$

where $\Delta S = S_{\text{weights}} - S_{\text{proto}} > 0$ is the round-trip bytes saved, $\Delta T_{\text{client}} = T_{\text{client,p}} - T_{\text{client,w}}$ the difference in non-overlapping client work and ΔT_{server} the corresponding server difference, since Eq. 5.5 gives $T_{\text{round,p}} - T_{\text{round,w}} = \Delta T_{\text{client}} + \Delta T_{\text{server}} - 8\Delta S/B$. Prototype exchange therefore completes a round sooner on any link slower than B^* when the prototype client does more work, and on every link when it does not. With the recorded saving of 1,542 Mbit per round and the measured client difference of Table 5.5, extraction plus exchange less the full-state exchange, 51.0 s on the Raspberry Pi 5 and 1.7 s on the Orin Nano, and with the server difference set to zero, prototype exchange completes a round sooner on links below 30 Mbit/s for the Pi client and below about 0.9 Gbit/s for the Orin client. Both figures compare the two exchanging arms. Against zero-exchange operation, the prototype client incurs the extraction pass whether or not it communicates.

Participation resilience. Controlled withholding asks whether useful collaboration continues when clients are intermittently unavailable. Three regimes are evaluated at the configuration of Table 5.2, each with three seeds, independent per-round absence of every client with probability 0.25 or 0.50, and a continuous 21-day

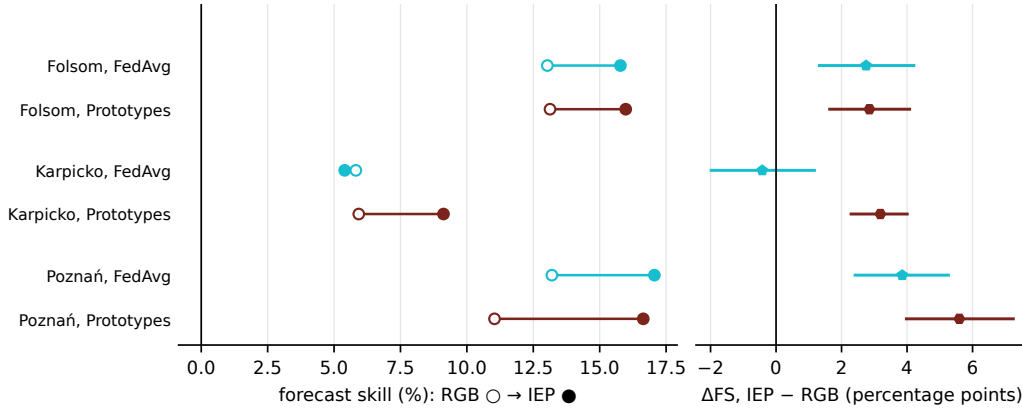


Fig. 5.5. Forecast skill of the two exchange mechanisms on the RGB (hollow) and physics-informed (IEP, filled) inputs at a pure-exchange operating point (daily synchronization, no warm-up, no replay, $C = 3$, $\lambda = 0.1$, squared-distance alignment), one seed per configuration (left), and the IEP-minus-RGB difference with its 95 % moving-block bootstrap interval over days (right). The static rows of Fig. 4.6 apply unchanged.

outage at Karpicko spanning three scheduled rounds. Aggregation runs whenever at least one client uploads, and since a single-uploader round returns only the client's own prototype (Sec. 5.2.1), the informative count is the number of rounds in which some client received a prototype from a different client. Under full participation, 51 of the 52 rounds deliver a peer prototype to some client, but only 28 have all three clients, because of Poznań's summer gap; under $p = 0.25$ the peer-delivering rounds fall to 36 to 42, under $p = 0.50$ to 18 to 24, and the outage removes Karpicko from three rounds and leaves 48. Figure 5.4 reports the outcomes against the same isolated reference. All twelve site-regime means remain above Isolated, and the nine target-site intervals exclude zero, whereas Folsom's never do; the target-site margins decline as withholding increases, Folsom changes little, consistent with its gain being self-regularization, and Karpicko's outage mean differs from its full-participation mean by less than the width of the interval. The margins therefore survive the loss of about half of the peer-delivering rounds at this operating point.

Physics-informed input under exchange. The seven-channel forecaster of Fig. 4.6 was also run through FedAvg and prototype exchange from its own checkpoint in a pure-exchange configuration with daily synchronization, no replay, $C = 3$, and the squared-distance alignment (Fig. 5.5). Prototype exchange on the physics-informed input reaches higher forecast skill than on the RGB input at all three sites, by 2.9, 3.2, and 5.6 points with intervals clear of zero. The input contrast is larger for prototype exchange than for FedAvg at the two target sites. Both prototype runs reach the angular concentration of the squared-distance variant by the final round yet stay ahead of FedAvg at two sites, so concentration alone does not set the ranking.

5.2.3 Conclusions

Regime-stratified prototype exchange keeps forecast skill at or above isolated local adaptation at all three sites, by 0.28 to 0.43 points, where each of the five parameter-exchange configurations loses 0.6 to 2.4 points to it. It does so with an exchanged object that is about a thousand times smaller when serialized. The ablations locate the design choices that matter. Aligning batch centroids with a contrastive objective keeps the representation spread, whereas per-sample squared-distance alignment concentrates it and costs 10 to 19 forecast skill points; twelve physically defined regimes are preferred to coarser partitions for their interpretability, since the partitions perform alike; and the aggregation weights are immaterial. The own-prototype control separates regularization from information. At the source site the gain is available without communication, at Karpicko peers add a resolved 0.3 points, and at Poznań the peer increment lies within the interval, so collaboration is justified by comparison with what the device can do alone and not by comparison with full-state exchange. Withholding half of the peer-delivering rounds leaves the target-site margins intact. On the devices, the compact message costs one extraction pass per day, 51 s on the Raspberry Pi 5 and 2 s on the Orin Nano, which makes a prototype round shorter than a full-state round on any link below 30 Mbit/s.

Limitations.

- The alignment weight was selected on the evaluation streams, the five parameter-exchange baselines run at one untuned setting each, and the ablation arms use two to three seeds. Hence, the small predictive margins are conditional on this selection, and the Folsom interval includes zero.
- The device profile times single clients whose returned objects are produced on the device, without transport or the coordinator, and the replay is season-aligned rather than contemporaneous; retention on the source partition, behavior at fleet scale, and exchange on top of the per-sample policy of Ch. 4 are not evaluated.

5.3 Summary

The chapter completes the progression of the dissertation from representation and local adaptation to the information available across devices. A locally adapted model responds to its own stream, and collaboration asks whether experience from another site can improve that response without transferring the raw observations or repeatedly replacing the whole model. The regime-stratified construction makes that exchange concrete for continuous irradiance forecasting. Prototype exchange and contrastive learning are established principles (Sec. 2.5.2); the regression-specific

regime construction, the own-versus-peer control, and the device profile are what this chapter contributes.

Answer to RQ4. Regime-stratified prototype exchange crosses the device interface with about a thousand times fewer values than full-state exchange, keeps forecast skill at or above isolated local adaptation at every site, where each of the five parameter-exchange configurations falls below it, and costs the device one extraction pass per day. Its peer increment over local adaptation is small and site-dependent, absent at the source site and resolved only at Karpicko (Sec. 5.2.3). The chapter therefore supports the communication component of the hypothesis and gives the decision rule that follows from it. Exchange is justified where a site's own stream leaves a resolved gap that peers close, and zero-exchange operation with own-prototype regularization is the preferred configuration elsewhere. Which of the two applies at a new site is a measurement to make at deployment, with the isolated learner and the payload record as the two references.

Conclusions

This dissertation asked whether four design strategies, task-specific sensing pipelines, hardware-aware execution, selective model updates, and compact knowledge exchange, can make sensing on edge devices efficient and reliable. Thermal images support occupancy counting and motor-state classification, and all-sky images with co-located irradiance measurements support forecasting, the one workload that runs through deployment, adaptation, and collaboration. Each choice was evaluated in the comparison designed for it, against the relevant reference methods, with predictive performance taken together with the measured cost of inference, adaptation or communication on the benchmark platforms of Table 2.6. That evidence, gathered along the whole lifecycle and reported with its cost, is the primary contribution of the dissertation. The nine criteria of Sec. 2.6.1 are the instrument that produced it. This chapter answers the four research questions, states the contributions, lists the limitations that qualify them, and sets out the work that follows.

6.1 Answers to the research questions

The central hypothesis of Sec. 1.2 is confirmed for each of its four components, in the sense defined in Sec. 2.7.1. Each component improved the trade-off between predictive performance and cost in the comparison designed for it, and Table 6.1 states the evidence and its scope for the hypothesis and for each objective. The paragraphs below give the answers themselves, while the numbers remain in the table and in the chapters it cites.

RQ1 and O1. Task-specific pipelines raise predictive performance at lower model capacity in all three workloads, and the domain knowledge that shapes them also serves to check what the model relies on, through attribution against fault physics in motor diagnosis and ablation of the physics-derived channels in forecasting. The same checks locate the limit of static validation. Robustness holds within a site across horizons, regimes, and resolutions, and it ends at relocation, where the forecaster validated at Folsom loses its forecast skill entirely at an unseen site without local data. That limit is the finding that motivates Chs. 4 and 5.

RQ2 and O2. Whether a configuration executes at all, and what an inference or a training step costs, are joint properties of architecture, numerical precision, and runtime, and none of the three can be inferred from the model alone. The same integer occupancy model reproduces its full-precision accuracy on one runtime and loses

Tab. 6.1. Assessment of the hypothesis and objectives O1 to O4. Contribution identifiers follow Sec. 6.2.

Claim	Status	Principal evidence and scope
H	Confirmed per component	Component gains in prediction, model size, execution cost, and payload (Tables 3.3, 4.13 and 5.4). The components were evaluated in separate studies on a shared framework; the proposed physics-grounded representation was reused in the adaptation and collaboration studies, but the full stack was not evaluated as one integrated deployed system.
O1 (C2)	Met within scope	Counting error halved with $3.55\times$ fewer parameters (Table 3.3). 3.83 forecast skill points from physics-informed channels in the ResNet50 ablation (Table 3.9); attribution consistent with fault physics for both fault classes, qualitative (Fig. 3.6); relocation without local data results in -34.31% forecast skill (Table 3.15).
O2 (C2, C4)	Met within scope	Feasibility and cost measured for twelve occupancy configurations and seven motor runtime paths (Tables 3.5, 3.6 and 3.8); accelerator-backed head training up to $15.4\times$ faster and 3.7 to $13.6\times$ less energy than the CPU (Table 4.13); update-time latency, energy, and peak memory measured for the forecasting policy (Table 4.6).
O3 (C3, C4)	Met within scope	Reduce-step policy above the static model at every site and level with sliding-window adaptation at 19 % to 48 % lower per-frame energy (Tables 4.3 and 4.6). Excluding detected persistent faults impedes adaptation with valid labels (Tables 4.8 and 4.9).
O4 (C5)	Met within scope	$980\times$ fewer exchanged values than full-state exchange, a serialized payload $946\times$ smaller (Table 5.4); forecast skill 0.28 to 0.43 points over isolated adaptation where parameter exchange loses 0.6 to 2.4 (Table 5.2); peer increment resolved at Karpicko (Table 5.3); margins continue under withholding (Fig. 5.4); client cost one extraction pass per day (Table 5.5).

part of it on another, conversion failures remove competitive motor architectures from at least one execution path and change the non-dominated set between the CPU and the accelerator, and under duty-cycled sampling the daily energy of a board is set by its idle draw rather than by the model, which reverses the platform ranking by energy per inference. For on-device training, a frozen integer backbone on an inference accelerator with a trainable head on the host gives the largest reduction in step time and energy of any tested configuration where the backbone tolerates integer features. Feasibility and cost therefore have to be measured on the target toolchain before architecture selection, not after it.

RQ3 and O3. A selective update policy under delayed supervision improves on static inference at every evaluated site, over recorded streams of 213 to 362 scored days, with the largest gains at the two sites the model had never seen. Reducing the encoder step makes the policy safe, since the original full-encoder action deteriorates

within months at the source site, and the thresholds set an FS–compute trade-off, not a fragile optimum. Against sliding-window adaptation the policy is level on forecast skill and slightly better on campaign error, at lower per-frame energy and with a daily update budget that fits inside the idle time of the acquisition period, and the storage proxy translates the gain into lower battery cycling. The admissibility study adds that a detected image fault is not by itself a reason to withhold the frame, because the tested persistent faults remained useful evidence with valid labels.

RQ4 and O4. Knowledge sharing preserves predictive performance relative to isolated local adaptation and adds to it where a site’s own stream leaves a gap. Regime-stratified prototype exchange keeps forecast skill at or above isolated adaptation at all three sites, whereas every parameter-exchange configuration loses forecast skill, and it does so with about a thousand times fewer exchanged values than the model state. The own-prototype control separates regularization from information, since at the source site the whole gain is available without communication, at Karpicko peers add a resolved peer increment, and at Poznań the peer increment lies within the interval. On the devices the message costs one extraction pass per day, which makes a prototype round shorter than a full-state round on any link below about 30 Mbit/s for the Raspberry Pi client. Exchange is therefore justified where peers close a resolved gap, and zero-exchange operation with own-prototype regularization is the operating point elsewhere.

6.2 Contributions

The five contributions below state what was built and why it transfers beyond the tested configurations. Contributions C1 through C5 cover objectives O1 through O4, and Table 6.1 holds the evidence for each.

- **C1. A reporting framework for reliable edge sensing.** (O1–O4.) Nine reliability criteria, three per lifecycle stage (Sec. 2.6.1), a shared set of reference platforms and metric definitions (Sec. 2.7), and an explicit reference for every gain, whether the static model, isolated local adaptation, or full-state exchange. The framework is reusable for any edge sensing study that reports predictive results with their measured cost, and Table 6.1 is its application here.
- **C2. Task-specific visual representations and compact deployment.** (O1, O2.) A density-regression occupancy pipeline that halves the counting error of the stronger published baseline with a model small enough for microcontrollers, physics-informed all-sky input channels that raise forecast skill beyond reimplemented published forecasters, and a thermal motor classifier whose attribution agrees with the fault physics. The design rule that transfers is to allocate model capacity to the structure of the task rather than to generic

depth, and to use the same domain knowledge as a check. Evidence in rows O1 and O2 of Table 6.1.

- **C3. Error-triggered adaptation and its limits.** (O3.) An error-triggered policy under delayed supervision whose realized error selects no update, a head update, or a reduced-step partial-encoder update, together with the ablation that attributes source-site safety to the reduction of the encoder step and the finding that a detected image fault does not by itself justify withholding a frame from learning. The transferable rules are to update on realized error, to keep the encoder step an order of magnitude below the head step, and to judge exclusion by its effect on learning. Evidence in row O3.
- **C4. Hardware-aware execution and measured failure modes.** (O2, O3.) The measurement, across runtimes, precisions, and devices, of which visual pipelines execute and what they cost, and a Hailo and ORT implementation of the frozen-integer-backbone, trainable-head partition with a cost model for its speedup. What transfers is the method, conversion as a screening gate before architecture selection, daily energy as the deployment metric under duty cycling, and the partition as the means of training on an inference accelerator. Evidence in rows O2 and O3.
- **C5. Regime-stratified prototype exchange for continuous forecasting.** (O4.) Physically defined irradiance regimes summarize all-sky embeddings into prototypes, an attention-weighted mixture returns to each client, and centroid-contrastive alignment guides local regression (Sec. 5.2.1), with an own-prototype control that separates regularization from peer information and a matched client profile that quantifies the cost of the exchange. The mechanism transfers to any regression task whose operating regimes can be defined from physical covariates. Evidence in row O4.

6.3 Limitations

Four limitations qualify the conclusions. The evidence behind each remains with the results and the limitations of Chs. 3 to 5.

- **Compositional scope.** The four mechanisms were evaluated in separate studies, with two exceptions. The physics-informed forecaster of Sec. 3.4 was also run under the update policy (Fig. 4.6) and under prototype exchange (Fig. 5.5), and the image-fault monitor was evaluated in series with the update policy, on the Folsom stream only. The accelerator-backed execution path was evaluated on classification workloads and was not combined with the reduced-step encoder action or with the trainable-encoder alignment of Ch. 5. Retention on the source distribution after adaptation was not measured. The gains of the chapters are therefore component gains, and the reliability criteria of Sec. 2.6.1 are met one stage at a time.

- **Evidence scope.** The evidence rests on three workloads, on single training runs in Ch. 3 and most arms of Ch. 4, and on configurations selected on the streams used for reporting, the step-reduction factor and thresholds of the adaptation policy, and the alignment weight of the exchange. The synthetic faults are rendered at one site and in one year; the motor cross-validation cannot be confirmed as trial-disjoint because its fold assignment file is not preserved. The motor attribution check is qualitative, one image per class, so a quantitative acceptance rule for physical consistency, with region masks defined in advance, attribution mass per region over every test frame, and a model-randomization control, remains to be built. The crossed representation checks are single-seed with unmatched training augmentation.
- **Systems scope.** Findings are specific to the evaluated hardware and to the software and toolchain releases in use when each study was run. Absolute latency, energy, and post-conversion accuracy can therefore change from release to release. Within each study, every method ran on the same setup, and the platform measurements were used to verify and compare them, so those comparisons do not depend on the release even where the absolute figures would.
- **Operational sensitivity.** Post-deployment learning relies on thresholds, an alignment weight, a synchronization cadence, and budgets that a new site or season may require to be recalibrated; within the tested ranges they set accuracy against compute, and a pair of thresholds fixed from source validation errors performed within 0.3 points of the reference. No stability guarantee is given for the update law, and the collapse of the tested test-time adaptation variant shows why update behavior has to be monitored.

6.4 Future work

The limitations above define the work that follows.

- **One system, evaluated end to end.** Compose the monitor, the update policy, the accelerator-backed execution path, and prototype exchange in one implementation that accommodates the reduced-step encoder action and keeps the monitor's feature source fixed, and evaluate candidate and reference configurations on the same streams with inference, update, and exchange costs measured at consistent scopes and with predictive tolerances stated before the data are seen.
- **Representation together with adaptation and exchange.** Repeat the crossed checks of Figs. 4.6 and 5.5 with the reduced-step action, matched training procedures, several seeds, and an isolated physics-informed configuration, and score them on common masks for sky regimes and ramp intervals, to establish whether input structure and post-deployment learning add or interact.

- **A stability guarantee for the head-only action.** For the frozen-encoder, linear-head action, where the model is linear in the updated parameters, the conditions for a stable parameter error under delayed regressors would turn the dead zone, clipping, and subspace restriction from design choices into an explicit guarantee, with dead-zone normalized least-mean-squares or recursive least-squares updates as the starting point.
- **Peer value on the stronger local learner.** Test prototype exchange on top of the per-sample conditional policy, time the exchange over a real link and coordinator against the measured client profile, evaluate predictive preservation on an untouched period with a stated harm tolerance, and extend the withholding experiments to longer absences and to the recovery after them.
- **Physically grounded stratification and decision-relevant evaluation.** Replace the clear-sky-index regimes with measures grounded in cloud dynamics, and evaluate forecasts by their usefulness in grid and storage dispatch as well as by error metrics, so that the accuracy–cost trade-off is assessed against the decisions it is meant to guide.

6.5 Ethics statement

The research was conducted in line with applicable legal and institutional regulations, including the General Data Protection Regulation (GDPR) where relevant. Privacy by design guided low-resolution sensing, local processing, data minimization, retention periods, and access restrictions. The goal is reliable sensing that enables better system performance and more efficient resource use, not surveillance or high-impact automated decisions about individuals. Exchanging prototypes rather than raw data reduces direct exposure but still requires safeguards against memorization, reconstruction, and other information leakage.

6.6 Statement on the use of generative AI

Writing support tools were employed to check formatting, language, and consistency. Code-oriented assistance tools were used during software development and for code review. No experimental measurements were generated. The author remains responsible for the text, code, analyses, and conclusions presented in this dissertation.

6.7 Declaration of research artifacts availability

Table 6.2 summarizes the research artifacts for each study, including GitHub repositories containing source code and configuration files, as well as Zenodo records.

These records provide versioned, persistently identified snapshots of datasets and experimental artifacts, enabling the reported analyses to be reproduced.

Tab. 6.2. Repositories and persistent identifiers for research source code and artifacts.

Study	Resources
Occupancy estimation in low-resolution thermal images	Source code: https://github.com/PUTvision/thermo-hardware-benchmark
Non-invasive thermal-image-based induction-motor fault diagnosis	Source code: https://github.com/MatPiech/motor-fault-diagnosis Dataset: https://doi.org/10.5281/zenodo.8203070
Intra-hour solar irradiance forecasting using all-sky images	Source code: https://github.com/PUTvision/solar-irradiance Artifacts: https://doi.org/10.5281/zenodo.15792282
Conditional on-device adaptation, input-admissibility gating, and prototype exchange	Source code: https://github.com/MatPiech/continual-solar-irradiance
On-device adaptation with an Edge AI inference accelerator	Source code: https://github.com/MatPiech/accelerator-training Artifacts: https://doi.org/10.5281/zenodo.20439109

List of publications

This appendix lists the scientific publications that support and extend the main body of this thesis. These scientific works were carried out during the doctoral studies (2021–2026), and the bibliographic information originates from the Scientific Information System of the Poznań University of Technology¹. The scientific works are divided into two groups. The first group includes those related to the doctoral thesis. The second group includes co-authored publications not directly related to the dissertation. In both groups, the presented works are divided into scientific articles, chapters in monographs or conference proceedings, and abstracts. The sections are arranged chronologically, starting with the most recent. Table A.1 presents a quantitative summary of publications, distinguishing thesis-related works from all publications carried out as part of the doctoral studies (2021–2026).

Publications strongly related to the doctoral dissertation

1. Scientific articles:

- 1.1. M. Piechocki and M. Kraft. “A systematic synthesis of sky image enhancement techniques for ground-based solar irradiance forecasting”. In: *Applied Energy* 410 (2026), p. 127533. DOI: 10.1016/j.apenergy.2026.127533 [Impact Factor: 11.0; Ministry Points: 200]
- 1.2. M. Piechocki et al. “Unraveling Induction Motor State through Thermal Imaging and Edge Processing: A Step towards Explainable Fault Diagnosis”. In: *Eksploracja i Niezawodność – Maintenance and Reliability* 25.3 (2023), p. 170114. DOI: 10.17531/ein/170114 [Impact Factor: 2.2; Ministry Points: 200]
- 1.3. M. Piechocki et al. “Efficient People Counting in Thermal Images: The Benchmark of Resource-Constrained Hardware”. In: *IEEE Access* 10 (2022), pp. 124835–124847. DOI: 10.1109/ACCESS.2022.3225233 [Impact Factor: 3.9; Ministry Points: 100]

2. Chapters in monograph/conference proceedings:

- 2.1. M. Piechocki et al. *Empowering On-Device Model Adaptation with an Edge AI Inference Accelerator*. Accepted to IoT, Edge, and Mobile for Embedded Machine Learning Workshop (ECML PKDD). 2026. DOI: 10.48550/arXiv.2607.18101. arXiv: 2607.18101 [cs.LG] [Accepted for publication]
- 2.2. M. Piechocki et al. “On-Device Continual Adaptation for Reliable Solar Irradiance Forecasting”. In: *Advanced Concepts for Intelligent Vision Systems*.

¹<https://sin.put.poznan.pl/people/details/mateusz.piechocki>

- Springer Nature Switzerland, 2026, pp. 353–364. DOI: 10.1007/978-3-032-07343-3_28 [Ministry Points: 70]
- 2.3. M. Piechocki and M. Kraft. “Anomaly Detection in Ground-Based Sky Imagery”. In: *Progress in Polish Artificial Intelligence Research 6. 6th Polish Conference on Artificial Intelligence (PP-RAI'2025)*, 07–09.04.2025, Katowice, Poland. Uniwersytet Śląski w Katowicach, 2025, pp. 95–100. DOI: 10.31261/PN.4331 [Ministry Points: 20]
 - 2.4. M. Piechocki et al. “Fault Diagnosis in a Squirrel-Cage Induction Motor Using Thermal Imaging”. In: *Progress in Polish Artificial Intelligence Research 4*. Wydawnictwo Politechniki Łódzkiej, 2023, pp. 37–42. DOI: 10.34658/9788366741928 [Ministry Points: 20]
 - 2.5. P. Aszkowski and M. Piechocki. “Thermo Presence: The Low-resolution Thermal Image Dataset and Occupancy Detection Using Edge Devices”. In: *Proceedings of the 3rd Polish Conference on Artificial Intelligence (PP-RAI 2022)*, Gdynia, Poland. Gdynia Maritime University, 2022, pp. 49–52. URL: https://wydawnictwo.umg.edu.pl/pp-rai2022/pdfs/11_pp-rai-2022-094.pdf (visited on Sept. 15, 2026) [Ministry Points: 20]

Other publications co-authored during the PhD (2021–2026)

1. Scientific articles:

- 1.1. M. Kraft et al. “Visual Feedback System Supporting Robotic Manipulation of Hemp Plants”. In: *Journal of Natural Fibers* 22.1 (2025), p. 2454261. DOI: 10.1080/15440478.2025.2454261 [Impact Factor: 2.8; Ministry Points: 140]
- 1.2. D. Pieczyński et al. “A fast, lightweight deep learning vision pipeline for autonomous UAV landing support with added robustness”. In: *Engineering Applications of Artificial Intelligence* 131 (2024), p. 107864. DOI: 10.1016/j.engappai.2024.107864 [Impact Factor: 7.5; Ministry Points: 140]
- 1.3. B. Ptak et al. “On-Board Crowd Counting and Density Estimation Using Low Altitude Unmanned Aerial Vehicles—Looking beyond Beating the Benchmark”. In: *Remote Sensing* 14.10 (2022), p. 2288. DOI: 10.3390/rs14102288 [Impact Factor: 5.0; Ministry Points: 100]

2. Chapters in monograph/conference proceedings:

- 2.1. M. Piechocki and A. Capotondi. “Benchmarking the Cost of Adaptation: A System-Level Analysis of Test-Time Adaptation on Edge Devices”. In: *Proceedings of the 24th Annual International Conference on Mobile Systems, Applications and Services Workshops*. Association for Computing Machinery, 2026, pp. 385–391. DOI: 10.1145/3812836.3814791 [Ministry Points: 70]
- 2.2. W. Bałtruszewicz et al. “Robotics in Modern Agriculture Applications Implemented by Inwebit in Cooperation with Scientific and Research

Institutions—Robotisation of Hemp Growth and Apple Harvesting”. In: *2024 13th International Workshop on Robot Motion and Control (RoMoCo)*. 2024, pp. 117–123. DOI: 10.1109/RoMoCo60539.2024.10604416 [Ministry Points: 20]

Tab. A.1. Quantitative summary of publications published during the doctoral studies (2021–2026), based on the Scientific Information System of the Poznań University of Technology. Values are reported separately for dissertation-related outputs and, in parentheses, for the full publication set. The work “*Empowering On-Device Model Adaptation with an Edge AI Inference Accelerator*” has been accepted for publication but is not included because it had not been published at the time of dissertation submission.

Publication type	Impact Factor	Ministry Points	No. Works
Scientific articles	17.1 (32.4)	500 (880)	3 (6)
Conference proceedings	—	130 (220)	4 (6)
Total	17.1 (32.4)	630 (1100)	7 (12)

List of grants and awards

This appendix lists the research grants and awards received by the author during his doctoral studies (2021–2026). Items are listed in chronological order, beginning with the most recent.

1. Grants:

- 1.1. In 2026, the author received a computational grant (no. pl1221) for a project entitled “Intra-hour Solar Irradiance Forecasting”, funded by the Poznań Supercomputing and Networking Center (PSNC).
- 1.2. In 2025, as part of **dAIEDGE 1st Open Call**, the author received funding for a 7-month research and mobility program for a project entitled “*Continual learning acceleration with on-device training on resource-constrained hardware*” at the University of Modena and Reggio Emilia (grant agreement no. dAI1OC01). A network of excellence for distributed, trustworthy, efficient and scalable AI at the Edge (dAIEDGE) was funded under the European Union’s Horizon Europe Research and Innovation Programme (grant agreement number: 101120726).
- 1.3. In 2025, the author was awarded a research grant under the **PhDBoost** program for the implementation of the research project entitled “*Reliable and computing-efficient test-time adaptation for real-world scenarios*”. PhDBoost is a program for doctoral students at the Poznań University of Technology, supporting research related to the doctoral dissertation and the scientific development of doctoral students.
- 1.4. In 2025 and 2024, the author received a grant to participate in the Summer School “*Advanced Computer Architecture and Compilation for High-performance Embedded Systems (ACACES)*”, organized by the HiPEAC network (High Performance, Edge And Cloud computing) in Fiuggi, Italy.
- 1.5. In 2024, the author had an opportunity to complete a 3-month research internship in the High-Performance Real-Time Laboratory at the University of Modena and Reggio Emilia, funded as part of the **Mobility IV** competition (INPUTDoc project under the NAWA STER program). The research internship was supervised by Prof. Marko Bertogna. It focused on hardware-accelerated, on-device training.
- 1.6. In 2023, as part of the **Talenty Jutra** competition funded by Fundacja Empiria i Wiedza, the author received a grant for implementation of a research project entitled “*An intelligent solution for accurate short-term irradiance forecasting*”. This project resulted in the development of low-cost

prototypes of devices for measurement and intra-hour solar irradiance forecasting.

- 1.7. In 2022, under the **TERRINet Call 8** competition, the author was awarded a mobility scholarship to conduct research at the GRVC Robotics Laboratory in Seville, Spain. The aim of the project was to develop an onboard camera-based landing system for unmanned aerial vehicles. The European Robotics Research Infrastructures Network (TERRINet) project was funded under the Horizon 2020 program (grant agreement number: 730994).

2. Awards:

- 2.1. Best Paper Award for the presentation of the work entitled “*Enhancing Solar Irradiance Forecasts with On-Device Continual Adaptation*” at Excellence in Artificial Intelligence and Edge Computing Workshop co-located with the HiPEAC conference, January 20, 2025, Barcelona (Spain).
- 2.2. Distinction in the national competition Młodzi Mistrzowie, organized by the program and organizing committee of the Forum Teleinformatyki, for the work entitled “*System for accurate short-term solar irradiance forecasting*”.

List of research projects

The following section lists the research projects in which the author participated during his doctoral studies (2021–2026). Since 2022, he has also contributed to research activities supported by the Dean’s research grants (Scientific Activities of Young Researchers; Scientific Activities Research Potential).

1. In 2024, the author was involved in the PUT-ISS project, part of the larger LeopardISS project. He implemented and optimized the Visual SLAM algorithm for real-time use application on a resource-constrained, space-qualified embedded computer. In 2025, the system was delivered to the International Space Station and completed its task successfully as part of the IGNIS mission.
2. In 2023–2024, the author contributed to an R&D project entitled “RENERgetic: Community-empowered Sustainable Multi-Vector Energy Islands”. This project was funded by the European Union’s Horizon 2020 research and innovation program (grant agreement number: 957845). The author was primarily responsible for developing and evaluating computer vision and edge computing to optimize BMS and HVAC control.
3. In 2021–2023, the author actively participated in the R&D project entitled “Increasing the quality of fiber hemp seed by robotization” (project number: POIR.01.01.01-00-2271/20), financed by the National Centre for Research and Development (NCBR). The author developed an efficient computer vision system for the detection and segmentation of hemp stalks for an autonomous agricultural robot designed to improve the quality of fiber hemp seeds.

List of Figures

1.1	The edge-device perception lifecycle considered in this dissertation. . .	2
2.1	Encoder–head interface and its three uses.	20
3.1	Example from the Thermo Presence dataset.	34
3.2	U-Net for occupancy estimation in low-resolution thermal images. . . .	34
3.3	Latency against daily energy of the occupancy configurations.	39
3.4	Diagram of the laboratory test bench.	41
3.5	Motor accuracy, latency, and model size.	44
3.6	Qualitative attribution inspection of the motor classifier.	45
3.7	Spatio-temporal irradiance forecasting model.	48
3.8	Construction of the multi-channel input tensor.	48
3.9	Nineteen-backbone forecasting comparison.	52
3.10	Forecast trajectories under three sky conditions.	56
3.11	Time-domain simulation of storage firming.	58
4.1	Threshold-based conditional adaptation under lagged supervision. . . .	65
4.2	Sky images from the training and deployment sites.	67
4.3	Contrasts against the static and sliding-window references.	69
4.4	Monthly error and forecast skill relative to the static model.	70
4.5	Battery-power duration curves of the forecast errors.	74
4.6	Reduced and physics-informed input under conditional adaptation. . .	75
4.7	Synthetic sensing-path faults on real frames.	78
4.8	Highest-scoring real frames, ranks 1–18.	80
4.9	Adaptation through a persistent fault, with and without exclusion. . .	80
4.10	Execution partition of the heterogeneous adaptation pipeline.	84
4.11	ONNX-based toolchain for heterogeneous adaptation.	85
4.12	Test accuracy under five INT8 restoration strategies.	87
4.13	Training throughput against batch size on Oxford-IIIT Pet.	88
4.14	Time-to-accuracy convergence on the classification workloads.	88
5.1	Client-side prototype construction.	96
5.2	Change in forecast skill relative to isolated local adaptation.	101
5.3	Response of forecast skill to the alignment weight.	101
5.4	Forecast skill under controlled client dropout.	105
5.5	Exchange mechanisms on the RGB and physics-informed inputs. . . .	106

List of Tables

1.1	Research objectives, workloads, and evidence.	4
2.1	Edge device categories and their dominant constraints.	8
2.2	Adaptation method families.	15
2.3	Positioning by lifecycle stage.	21
2.4	Datasets and deployment streams.	23
2.5	Forecaster configurations used in the empirical chapters.	25
2.6	Reference edge platforms.	26
3.1	Evaluation protocol of the three deployment studies.	32
3.2	Thermo Presence sequence split.	33
3.3	Thermal counting quality and model complexity.	36
3.4	Counting performance by occupancy level.	36
3.5	Counting accuracy across deployed configurations.	37
3.6	Occupancy inference latency, power, and energy.	38
3.7	Motor-classification cross-validation scores.	42
3.8	Average inference time across edge runtimes.	43
3.9	ResNet50 loss and representation ablation.	51
3.10	Decomposition of the ResNet50 gain.	52
3.11	Reimplemented forecasting-method comparison.	53
3.12	Forecast horizon sweep on the Folsom test split.	54
3.13	Forecasting ability by input resolution.	54
3.14	Forecasting latency by processing stage.	55
3.15	Cross-site forecasting transfer.	57
4.1	Evaluation protocol of the three adaptation studies.	62
4.2	Notation used throughout this chapter.	63
4.3	Continual adaptation for 15-minute irradiance forecasting.	68
4.4	Ablation of the encoder action and sensitivity to the thresholds.	71
4.5	Recorded update actions, per-action latency and the resulting update budget.	72
4.6	Measured device cost of the adaptation strategies.	73
4.7	Storage duty implied by the forecast errors.	74
4.8	Detection of synthetic sensing faults.	79
4.9	Predictive outcomes under sensing faults.	81

4.10 Update counts and forecast-skill contrasts under sensing faults. 82

4.11 Frozen-backbone characterization at $3 \times 224 \times 224$ input resolution. . . 85

4.12 Performance restoration strategies and configurations. 86

4.13 End-to-end head-only fine-tuning results with batch size 1. 87

5.1 Collaborative-learning design space and exchanged state. 98

5.2 Forecasting performance of collaborative and local methods. 100

5.3 Component ablation of the proposed configuration. 102

5.4 Per-round communication payload and analytical value counts. 103

5.5 Per-client device cost of isolated, full-state and prototype clients. 104

6.1 Evidence status of the hypothesis and research objectives. 110

6.2 Repositories and persistent identifiers for research source code and artifacts. 115

A.1 Quantitative summary of publications (2021–2026). 119

Bibliography

- [1] R. Raj and A. Kos. “An Extensive Study of Convolutional Neural Networks: Applications in Computer Vision for Improved Robotics Perceptions”. In: *Sensors* 25.4 (2025), p. 1033. DOI: 10.3390/s25041033 (cit. on p. 1).
- [2] N. Hütten, M. Alves Gomes, F. Hölken, et al. “Deep Learning for Automated Visual Inspection in Manufacturing and Maintenance: A Survey of Open-Access Papers”. In: *Applied System Innovation* 7.1 (2024), p. 11. DOI: 10.3390/asi7010011 (cit. on p. 1).
- [3] S. Bhattacharya, S. R. K. Somayaji, T. R. Gadekallu, M. Alazab, and P. K. R. Maddikunta. “A review on deep learning for future smart cities”. In: *Internet Technology Letters* 5.1 (2022), e187. DOI: 10.1002/itl2.187 (cit. on p. 1).
- [4] S. S. Gill, M. Golec, J. Hu, et al. “Edge AI: A Taxonomy, Systematic Review and Future Directions”. In: *Cluster Computing* 28.1 (2024), p. 18. DOI: 10.1007/s10586-024-04686-y (cit. on pp. 1, 7).
- [5] F. Golpayegani, N. Chen, N. Afraz, et al. “Adaptation in Edge Computing: A Review on Design Principles and Research Challenges”. In: *ACM Trans. Auton. Adapt. Syst.* 19.3 (2024). DOI: 10.1145/3664200 (cit. on pp. 1, 7, 13).
- [6] B. Wu, Z. Ding, and J. Huang. “A Review of Continual Learning in Edge AI”. In: *IEEE Transactions on Network Science and Engineering* 13 (2026), pp. 6571–6588. DOI: 10.1109/TNSE.2026.3657652 (cit. on pp. 2, 16, 31).
- [7] D. Yu, Z. Xie, Y. Yuan, et al. “Trustworthy decentralized collaborative learning for edge intelligence: A survey”. In: *High-Confidence Computing* 3.3 (2023), p. 100150. DOI: 10.1016/j.hcc.2023.100150 (cit. on pp. 2, 13).
- [8] S.-A. Logothetis, V. Salamalikis, B. Nouri, et al. “Solar Irradiance Ramp Forecasting Based on All-Sky Imagers”. In: *Energies* 15.17 (2022), p. 6191. DOI: 10.3390/en15176191 (cit. on pp. 5, 47).
- [9] G. Lin, A. Casillas, M. Sheng, and J. Granderson. “Performance Evaluation of an Occupancy-Based HVAC Control System in an Office Building”. In: *Energies* 16.20 (2023), p. 7088. DOI: 10.3390/en16207088 (cit. on pp. 5, 33).
- [10] K. Hamani, M. Kuchar, M. Kubatko, and S. Kirschner. “Advancements in Induction Motor Fault Diagnosis and Condition Monitoring: A Comprehensive Review”. In: *Sensors* 25.19 (2025), p. 5942. DOI: 10.3390/s25195942 (cit. on pp. 5, 40).

- [11] M. Piechocki, M. Kraft, T. Pajchrowski, P. Aszkowski, and D. Pieczyński. “Efficient People Counting in Thermal Images: The Benchmark of Resource-Constrained Hardware”. In: *IEEE Access* 10 (2022), pp. 124835–124847. DOI: 10.1109/ACCESS.2022.3225233 (cit. on pp. 6, 31, 35–38, 117).
- [12] P. Aszkowski and M. Piechocki. “Thermo Presence: The Low-resolution Thermal Image Dataset and Occupancy Detection Using Edge Devices”. In: *Proceedings of the 3rd Polish Conference on Artificial Intelligence (PP-RAI 2022), Gdynia, Poland*. Gdynia Maritime University, 2022, pp. 49–52. URL: https://wydawnictwo.umg.edu.pl/pp-rai2022/pdfs/11_pp-rai-2022-094.pdf (visited on Sept. 15, 2026) (cit. on pp. 6, 22, 31, 118).
- [13] M. Piechocki, T. Pajchrowski, M. Kraft, M. Wolkiewicz, and P. Ewert. “Unraveling Induction Motor State through Thermal Imaging and Edge Processing: A Step towards Explainable Fault Diagnosis”. In: *Eksploracja i Niezawodność – Maintenance and Reliability* 25.3 (2023), p. 170114. DOI: 10.17531/ein/170114 (cit. on pp. 6, 23, 31, 41–43, 117).
- [14] M. Piechocki, M. Kraft, and T. Pajchrowski. “Fault Diagnosis in a Squirrel-Cage Induction Motor Using Thermal Imaging”. In: *Progress in Polish Artificial Intelligence Research 4*. Wydawnictwo Politechniki Łódzkiej, 2023, pp. 37–42. DOI: 10.34658/9788366741928 (cit. on pp. 6, 23, 31, 118).
- [15] M. Piechocki and M. Kraft. “A systematic synthesis of sky image enhancement techniques for ground-based solar irradiance forecasting”. In: *Applied Energy* 410 (2026), p. 127533. DOI: 10.1016/j.apenergy.2026.127533 (cit. on pp. 6, 24, 31, 49, 57, 117).
- [16] M. Piechocki and M. Kraft. “Anomaly Detection in Ground-Based Sky Imagery”. In: *Progress in Polish Artificial Intelligence Research 6. 6th Polish Conference on Artificial Intelligence (PP-RAI’2025), 07–09.04.2025, Katowice, Poland*. Uniwersytet Śląski w Katowicach, 2025, pp. 95–100. DOI: 10.31261/PN.4331 (cit. on pp. 6, 61, 76, 118).
- [17] M. Piechocki, M. Kraft, and A. Capotondi. “On-Device Continual Adaptation for Reliable Solar Irradiance Forecasting”. In: *Advanced Concepts for Intelligent Vision Systems*. Springer Nature Switzerland, 2026, pp. 353–364. DOI: 10.1007/978-3-032-07343-3_28 (cit. on pp. 6, 61, 66, 117).
- [18] M. Piechocki, A. Capotondi, and M. Kraft. *Empowering On-Device Model Adaptation with an Edge AI Inference Accelerator*. Accepted to IoT, Edge, and Mobile for Embedded Machine Learning Workshop (ECML PKDD). 2026. DOI: 10.48550/arXiv.2607.18101. arXiv: 2607.18101 [cs.LG] (cit. on pp. 6, 61, 86, 89, 117).
- [19] J. Pérez, J. Díaz, J. Berrocal, R. López-Viana, and Á. González-Prieto. “Edge computing”. In: *Computing* 104.12 (2022), pp. 2711–2747. DOI: 10.1007/s00607-022-01104-2 (cit. on p. 7).
- [20] H. Hua, Y. Li, T. Wang, et al. “Edge Computing with Artificial Intelligence: A Machine Learning Perspective”. In: *ACM Computing Surveys* 55.9 (2023). DOI: 10.1145/3555802 (cit. on p. 7).
- [21] M. Sharma, A. Tomar, and A. Hazra. “Edge Computing for Industry 5.0: Fundamental, Applications, and Research Challenges”. In: *IEEE Internet of Things Journal* 11.11 (2024), pp. 19070–19093. DOI: 10.1109/JIOT.2024.3359297 (cit. on p. 7).

- [22] R. David, J. Duke, A. Jain, et al. “TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems”. In: *Proceedings of Machine Learning and Systems*. Vol. 3. 2021, pp. 800–811. URL: https://proceedings.mlsys.org/paper_files/paper/2021/file/6c44dc73014d66ba49b28d483a8f8b0d-Paper.pdf (visited on Sept. 15, 2026) (cit. on pp. 7, 10, 21).
- [23] Y. Abadade, A. Temouden, H. Bamoumen, et al. “A Comprehensive Survey on TinyML”. In: *IEEE Access* 11 (2023), pp. 96892–96922. DOI: 10.1109/ACCESS.2023.3294111 (cit. on p. 7).
- [24] S. Somvanshi, M. M. Islam, G. Chhetri, et al. “From Tiny Machine Learning to Tiny Deep Learning: A Survey”. In: *ACM Computing Surveys* 58.7 (2025). DOI: 10.1145/3776588 (cit. on p. 8).
- [25] D. Ngo, H.-C. Park, and B. Kang. “Edge Intelligence: A Review of Deep Neural Network Inference in Resource-Limited Environments”. In: *Electronics* 14.12 (2025), p. 2495. DOI: 10.3390/electronics14122495 (cit. on p. 9).
- [26] Y. Zhu, M. Wang, X. Yin, et al. “Deep Learning in Diverse Intelligent Sensor Based Systems”. In: *Sensors* 23.1 (2023), p. 62. DOI: 10.3390/s23010062 (cit. on p. 9).
- [27] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on pp. 9, 42, 47, 48, 50, 85).
- [28] A. Howard, M. Sandler, B. Chen, et al. “Searching for MobileNetV3”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 1314–1324. DOI: 10.1109/ICCV.2019.00140 (cit. on pp. 9, 85).
- [29] D. Qin, C. Leichner, M. Delakis, et al. “MobileNetV4: Universal Models for the Mobile Ecosystem”. In: *Computer Vision – ECCV 2024*. Cham: Springer Nature Switzerland, 2025, pp. 78–96. DOI: 10.1007/978-3-031-73661-2_5 (cit. on p. 9).
- [30] M. Tan and Q. Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6105–6114. URL: <https://proceedings.mlr.press/v97/tan19a.html> (visited on Sept. 15, 2026) (cit. on pp. 9, 42, 50, 85).
- [31] P. K. Anasosalu Vasu, J. Gabriel, J. Zhu, O. Tuzel, and A. Ranjan. “FastViT: A Fast Hybrid Vision Transformer using Structural Reparameterization”. In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023, pp. 5762–5772. DOI: 10.1109/ICCV51070.2023.00532 (cit. on pp. 10, 85).
- [32] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, and S. Han. “Tiny Machine Learning: Progress and Futures [Feature]”. In: *IEEE Circuits and Systems Magazine* 23.3 (2023), pp. 8–34. DOI: 10.1109/MCAS.2023.3302182 (cit. on p. 10).
- [33] J. Yang, Z. Li, Z. Feng, and Y. Xie. “A Survey on Neural Network Quantization”. In: *Proceedings of the 2025 6th International Conference on Computer Information and Big Data Applications*. Association for Computing Machinery, 2025, pp. 384–394. DOI: 10.1145/3746709.3746773 (cit. on p. 10).

- [34] M. Gholizade, H. Soltanizadeh, M. Rahmanimanesh, and S. S. Sana. “A review of recent advances and strategies in transfer learning”. In: *International Journal of System Assurance Engineering and Management* 16.3 (2025), pp. 1123–1162. DOI: 10.1007/s13198-024-02684-2 (cit. on p. 10).
- [35] Y. D. Kwon, R. Li, S. Venieris, et al. “TinyTrain: Resource-Aware Task-Adaptive Sparse Training of DNNs at the Data-Scarce Edge”. In: *Proceedings of the 41st International Conference on Machine Learning*. Vol. 235. Proceedings of Machine Learning Research. PMLR, 21–27 Jul 2024, pp. 25812–25843. URL: <https://proceedings.mlr.press/v235/kwon24c.html> (visited on Sept. 15, 2026) (cit. on pp. 10, 16, 20, 21).
- [36] M. Hussain, M. O’Nils, J. Lundgren, and S. J. Mousavirad. “A Comprehensive Review on Deep Learning-Based Data Fusion”. In: *IEEE Access* 12 (2024), pp. 180093–180124. DOI: 10.1109/ACCESS.2024.3508271 (cit. on p. 10).
- [37] F. Zhao, C. Zhang, and B. Geng. “Deep Multimodal Data Fusion”. In: *ACM Computing Surveys* 56.9 (2024). DOI: 10.1145/3649447 (cit. on p. 10).
- [38] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar. “Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems”. In: *ACM Computing Surveys* 55.4 (2022). DOI: 10.1145/3514228 (cit. on p. 10).
- [39] R. Wang and R. Yu. “Physics-Guided Deep Learning for Dynamical Systems: A Survey”. In: *ACM Computing Surveys* 58.5 (2025). DOI: 10.1145/3766887 (cit. on p. 10).
- [40] Microsoft. *ONNX Runtime*. URL: <https://onnxruntime.ai> (visited on Sept. 15, 2026) (cit. on pp. 11, 42, 86).
- [41] Google. *tf-lite-runtime 2.11.0*. 2022. URL: <https://www.tensorflow.org/lite> (visited on Sept. 15, 2026) (cit. on pp. 11, 42).
- [42] Google. *TensorFlow Lite for Microcontrollers*. URL: <https://www.tensorflow.org/lite/microcontrollers> (visited on Sept. 15, 2026) (cit. on p. 11).
- [43] E. Meller, A. Finkelstein, U. Almog, and M. Grobman. “Same, Same But Different: Recovering Neural Network Quantization Error Through Weight Factorization”. In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 4486–4495. URL: <https://proceedings.mlr.press/v97/meller19a.html> (visited on Sept. 15, 2026) (cit. on pp. 12, 21, 86).
- [44] A. Finkelstein, U. Almog, and M. Grobman. *Fighting quantization bias with bias*. 2019. DOI: 10.48550/arXiv.1906.03193. arXiv: 1906.03193 (cit. on pp. 12, 86).
- [45] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort. “Up or Down? Adaptive Rounding for Post-Training Quantization”. In: *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 2020, pp. 7197–7206. URL: <https://proceedings.mlr.press/v119/nagel20a.html> (cit. on pp. 12, 21, 86).
- [46] A. Finkelstein, E. Fuchs, I. Tal, et al. “QFT: Post-training Quantization via Fast Joint Finetuning of All Degrees of Freedom”. In: *Computer Vision – ECCV 2022 Workshops*. Springer Nature Switzerland, 2023, pp. 115–129. DOI: 10.1007/978-3-031-25082-8_8. (Visited on Sept. 15, 2026) (cit. on pp. 12, 86).

- [47] F. Friedrich, W. Stammer, P. Schramowski, and K. Kersting. “A typology for exploring the mitigation of shortcut behaviour”. In: *Nature Machine Intelligence* 5.3 (2023), pp. 319–330. DOI: 10.1038/s42256-023-00612-w (cit. on p. 12).
- [48] Y. Zhou, J. Zhang, Q. Cao, and J. Hu. “A first look at temporal bias in industrial image datasets”. In: *Neurocomputing* 655 (2025), p. 131488. DOI: 10.1016/j.neucom.2025.131488 (cit. on p. 12).
- [49] J. Adebayo, J. Gilmer, M. Muelly, et al. “Sanity checks for saliency maps”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems. NIPS’18*. Montréal, Canada: Curran Associates Inc., 2018, pp. 9525–9536. URL: <https://dl.acm.org/doi/10.5555/3327546.3327621> (visited on Sept. 15, 2026) (cit. on pp. 12, 46).
- [50] L. Longo, M. Brcic, F. Cabitza, et al. “Explainable Artificial Intelligence (XAI) 2.0: A manifesto of open challenges and interdisciplinary research directions”. In: *Information Fusion* 106 (2024), p. 102301. DOI: 10.1016/j.inffus.2024.102301 (cit. on pp. 12, 21).
- [51] M. Skliarov, R. El Shawi, C. Dhaoui, and N. Ahmed. “A comparative evaluation of explainability techniques for image data”. In: *Scientific Reports* 15.1 (2025), p. 41898. DOI: 10.1038/s41598-025-25839-y (cit. on pp. 12, 21).
- [52] Y. Sun, Y. Ming, X. Zhu, and Y. Li. “Out-of-Distribution Detection with Deep Nearest Neighbors”. In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 20827–20840. URL: <https://proceedings.mlr.press/v162/sun22d.html> (visited on Sept. 15, 2026) (cit. on p. 13).
- [53] F. T. Liu, K. M. Ting, and Z.-H. Zhou. “Isolation-Based Anomaly Detection”. In: *ACM Transactions on Knowledge Discovery from Data* 6.1 (2012), 3:1–3:39. DOI: 10.1145/2133360.2133363 (cit. on pp. 13, 77).
- [54] D. Wang, E. Shelhamer, S. Liu, B. Olshausen, and T. Darrell. “Tent: Fully Test-Time Adaptation by Entropy Minimization”. In: *International Conference on Learning Representations*. 2021. URL: <https://openreview.net/forum?id=uX13bZLkr3c> (visited on Sept. 15, 2026) (cit. on pp. 15, 21).
- [55] M. Zhang, S. Levine, and C. Finn. “MEMO: Test Time Robustness via Adaptation and Augmentation”. In: *Advances in Neural Information Processing Systems*. Vol. 35. Curran Associates, Inc., 2022, pp. 38629–38642. DOI: 10.52202/068431-2799 (cit. on pp. 15, 21, 66, 73).
- [56] T. Gong, J. Jeong, T. Kim, et al. “NOTE: Robust Continual Test-time Adaptation Against Temporal Correlation”. In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 27253–27266. DOI: 10.52202/068431-1976 (cit. on pp. 15, 21, 66, 73).
- [57] Q. Wang, O. Fink, L. Van Gool, and D. Dai. “Continual Test-Time Domain Adaptation”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 7191–7201. DOI: 10.1109/CVPR52688.2022.00706 (cit. on pp. 15, 21).
- [58] S. Zhu, T. Voigt, F. Rahimian, and J. Ko. “On-device Training: A First Overview on Existing Systems”. In: *ACM Trans. Sen. Netw.* 20.6 (2024). DOI: 10.1145/3696003 (cit. on pp. 15, 16).

- [59] J. Lin, L. Zhu, W.-M. Chen, et al. “On-Device Training Under 256KB Memory”. In: *Advances in Neural Information Processing Systems*. Vol. 35. Curran Associates, Inc., 2022, pp. 22941–22954. DOI: 10.52202/068431-1667 (cit. on pp. 15, 20, 21).
- [60] S. Arora, R. Rani, and N. Saxena. “A systematic review on detection and adaptation of concept drift in streaming data using machine learning techniques”. In: *WIREs Data Mining and Knowledge Discovery* 14.4 (2024), e1536. DOI: 10.1002/widm.1536 (cit. on p. 16).
- [61] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas. “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Vol. 54. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1273–1282. URL: <https://proceedings.mlr.press/v54/mcmahan17a.html> (visited on Sept. 15, 2026) (cit. on pp. 17, 20, 21, 93, 97).
- [62] P. Kairouz and H. B. McMahan. “Advances and Open Problems in Federated Learning”. In: *Foundations and Trends in Machine Learning* 14.1–2 (2021), pp. 1–210. DOI: 10.1561/22000000083 (cit. on p. 17).
- [63] Y. Zhao, M. Li, L. Lai, et al. *Federated Learning with Non-IID Data*. 2018. DOI: 10.48550/arXiv.1806.00582. arXiv: 1806.00582 [cs.LG] (cit. on p. 17).
- [64] T. Li, A. K. Sahu, M. Zaheer, et al. “Federated Optimization in Heterogeneous Networks”. In: *Proceedings of Machine Learning and Systems*. Vol. 2. 2020, pp. 429–450. URL: https://proceedings.mlsys.org/paper_files/paper/2020/file/1f5fe83998a09396ebe6477d9475ba0c-Paper.pdf (cit. on pp. 17, 21, 98).
- [65] X. Li, M. Jiang, X. Zhang, M. Kamp, and Q. Dou. “FedBN: Federated Learning on Non-IID Features via Local Batch Normalization”. In: *International Conference on Learning Representations (ICLR)*. 2021. URL: <https://openreview.net/forum?id=6YEQUn0QICG> (visited on Sept. 15, 2026) (cit. on pp. 17, 21, 98).
- [66] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary. *Federated Learning with Personalization Layers*. 2019. DOI: 10.48550/arXiv.1912.00818. arXiv: 1912.00818 [cs.LG] (cit. on pp. 17, 98).
- [67] W.-B. Kou, G. Zhu, B. Cheng, et al. “FedEMA: Federated Exponential Moving Averaging with Negative Entropy Regularizer in Autonomous Driving”. In: *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2025, pp. 10190–10197. DOI: 10.1109/IROS60139.2025.11246711 (cit. on pp. 18, 98).
- [68] Y. Tan, G. Long, L. Liu, et al. “FedProto: Federated Prototype Learning across Heterogeneous Clients”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 8. 2022, pp. 8432–8440. DOI: 10.1609/aaai.v36i8.20819 (cit. on pp. 18, 20, 21, 95).
- [69] W. Huang, M. Ye, Z. Shi, H. Li, and B. Du. “Rethinking Federated Learning with Domain Shift: A Prototype View”. In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, pp. 16312–16322. DOI: 10.1109/CVPR52729.2023.01565 (cit. on pp. 18, 20, 21).
- [70] H. Gong, L. Jiang, X. Liu, et al. “Gradient leakage attacks in federated learning”. In: *Artificial Intelligence Review* 56.1 (Oct. 2023), pp. 1337–1374. DOI: 10.1007/s10462-023-10550-z (cit. on pp. 18, 94).

- [71] K. Bonawitz, V. Ivanov, B. Kreuter, et al. “Practical Secure Aggregation for Privacy-Preserving Machine Learning”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1175–1191. DOI: 10.1145/3133956.3133982 (cit. on pp. 18, 94).
- [72] M. Abadi, A. Chu, I. Goodfellow, et al. “Deep Learning with Differential Privacy”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 308–318. DOI: 10.1145/2976749.2978318 (cit. on pp. 18, 94).
- [73] C. Banbury, V. Janapa Reddi, P. Torelli, et al. “MLPerf Tiny Benchmark”. In: *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Vol. 1. 2021. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/da4fb5c6e93e74d3df8527599fa62642-Paper-round1.pdf (visited on Sept. 15, 2026) (cit. on pp. 21, 32).
- [74] J. Zhang, Y. Xu, J. Xue, and W. Xiao. “Real-time prediction of solar radiation based on online sequential extreme learning machine”. In: *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*. 2018, pp. 53–57. DOI: 10.1109/ICIEA.2018.8397688 (cit. on pp. 21, 67, 68, 73).
- [75] P. Lara-Benítez, M. Carranza-García, J. M. Luna-Romera, and J. C. Riquelme. “Short-term solar irradiance forecasting in streaming with deep learning”. In: *Neurocomputing* 546 (2023), p. 126312. DOI: 10.1016/j.neucom.2023.126312 (cit. on pp. 21, 67–69, 72, 73).
- [76] M. Kraft, P. Aszkowski, D. Pieczyński, and M. Fularz. “Low-Cost Thermal Camera-Based Counting Occupancy Meter Facilitating Energy Saving in Smart Buildings”. In: *Energies* 14.15 (2021), p. 4542. DOI: 10.3390/en14154542 (cit. on pp. 22, 23, 33, 36).
- [77] H. T. C. Pedro, D. P. Larson, and C. F. M. Coimbra. *A comprehensive dataset for the accelerated development and benchmarking of solar forecasting methods*. Dataset (same title as the associated journal article). Zenodo, June 2019. DOI: 10.5281/zenodo.2826939. (Visited on Sept. 15, 2026) (cit. on pp. 23, 24, 47).
- [78] H. T. C. Pedro, D. P. Larson, and C. F. M. Coimbra. “A comprehensive dataset for the accelerated development and benchmarking of solar forecasting methods”. In: *Journal of Renewable and Sustainable Energy* 11.3 (2019), p. 036102. DOI: 10.1063/1.5094494 (cit. on pp. 23, 24, 47).
- [79] E. A. Papatheofanous, V. Kalekis, G. Venitourakis, F. Tziolos, and D. Reisis. “Deep Learning-Based Image Regression for Short-Term Solar Irradiance Forecasting on the Edge”. In: *Electronics* 11.22 (2022), p. 3794. DOI: 10.3390/electronics11223794 (cit. on pp. 24, 48, 53).
- [80] *Raspberry Pi 4 Model B: Specifications*. Raspberry Pi Foundation. 2019. URL: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications> (visited on Sept. 15, 2026) (cit. on pp. 26, 42).
- [81] *Raspberry Pi 5: Product Brief*. Rev. 1. Raspberry Pi Ltd. Aug. 2024. URL: <https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf> (visited on Sept. 15, 2026) (cit. on pp. 26, 86).

- [82] Intel Corporation. *Intel Neural Compute Stick 2*. 2019. URL: <https://www.intel.com/content/www/us/en/developer/articles/tool/neural-compute-stick.html> (visited on Sept. 15, 2026) (cit. on pp. 26, 42).
- [83] *Hailo-8L M.2 AI Acceleration Module: Product Brief*. Rev. 1. HAILO. Mar. 2024. URL: <https://hailo.ai/files/hailo-8l-m-2-et-product-brief-en/> (visited on Sept. 15, 2026) (cit. on pp. 26, 86).
- [84] *NVIDIA Jetson Orin Nano Super Developer Kit: Datasheet*. Rev. 2. NVIDIA Corporation. Dec. 2024. URL: <https://nvdam.widen.net/s/zkfqjmtds2/jetson-orin-datasheet-nano-developer-kit-3575392-r2> (visited on Sept. 15, 2026) (cit. on pp. 26, 86).
- [85] P. Ineichen and R. Perez. “A new airmass independent formulation for the Linke turbidity coefficient”. In: *Solar Energy* 73.3 (2002), pp. 151–157. DOI: 10.1016/S0038-092X(02)00045-2 (cit. on p. 28).
- [86] M. Satyanarayanan. “The Emergence of Edge Computing”. In: *Computer* 50.1 (2017), pp. 30–39. DOI: 10.1109/MC.2017.9 (cit. on pp. 31, 61).
- [87] *MLX90640 32x24 IR Array Datasheet*. 3901090640. Rev. 12. Melexis. Dec. 2019. URL: <https://media.melexis.com/-/media/files/documents/datasheets/mlx90640-datasheet-melexis.pdf> (visited on Sept. 15, 2026) (cit. on p. 33).
- [88] A. Metwaly, J. P. Queralta, V. K. Sarker, et al. “Edge computing with embedded AI: Thermal image analysis for occupancy estimation in intelligent buildings”. In: *Proceedings of the INTelligent Embedded Systems Architectures and Applications Workshop 2019*. 2019, pp. 1–6. DOI: 10.1145/3372394.3372397 (cit. on pp. 33, 36).
- [89] O. Ronneberger, P. Fischer, and T. Brox. “U-Net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28 (cit. on p. 35).
- [90] R. Wang, X. Zhan, H. Bai, et al. “A Review of Fault Diagnosis Methods for Rotating Machinery Using Infrared Thermography”. In: *Micromachines* 13.10 (2022), p. 1644. DOI: 10.3390/mi13101644 (cit. on pp. 40, 41).
- [91] S. Halder, S. Bhat, D. Zychma, and P. Sowa. “Broken Rotor Bar Fault Diagnosis Techniques Based on Motor Current Signature Analysis for Induction Motor - A Review”. In: *Energies* 15.22 (2022), p. 8569. DOI: 10.3390/en15228569 (cit. on p. 40).
- [92] P. Gangsar and R. Tiwari. “Signal based condition monitoring techniques for fault detection and diagnosis of induction motors: A state-of-the-art review”. In: *Mechanical Systems and Signal Processing* 144 (2020), p. 106908. DOI: 10.1016/j.ymssp.2020.106908 (cit. on p. 40).
- [93] Workswell. *Workswell InfraRed Camera (WIC) - stationary radiometric thermal camera*. 2020. URL: <https://workswell-thermal-camera.com/workswell-infrared-camera-wic> (visited on Sept. 15, 2026) (cit. on p. 40).
- [94] O. Janssens, R. Schulz, V. Slavkovikj, et al. “Thermal image based fault diagnosis for rotating machinery”. In: *Infrared Physics & Technology* 73 (2015), pp. 78–87. DOI: 10.1016/j.infrared.2015.09.004 (cit. on p. 41).

- [95] G. Singh, T. Anil Kumar, and V. Naikan. “Induction motor inter turn fault detection using infrared thermographic analysis”. In: *Infrared Physics & Technology* 77 (2016), pp. 277–282. DOI: 10.1016/j.infrared.2016.06.010 (cit. on p. 41).
- [96] I. Loshchilov and F. Hutter. “Decoupled Weight Decay Regularization”. In: *International Conference on Learning Representations*. 2019. DOI: 10.48550/arXiv.1711.05101 (cit. on p. 41).
- [97] C. Cui, T. Gao, S. Wei, et al. *PP-LCNet: A Lightweight CPU Convolutional Neural Network*. 2021. DOI: 10.48550/arXiv.2109.15099. arXiv: 2109.15099 [cs.CV] (cit. on p. 42).
- [98] M. Tan, B. Chen, R. Pang, et al. “MnasNet: Platform-Aware Neural Architecture Search for Mobile”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 2815–2823. DOI: 10.1109/CVPR.2019.00293 (cit. on p. 42).
- [99] M. Tan and Q. V. Le. “MixConv: Mixed Depthwise Convolutional Kernels”. In: *British Machine Vision Conference (BMVC)*. 2019. DOI: 10.48550/arXiv.1907.09595 (cit. on p. 42).
- [100] Arm. *Arm NN 23.02*. 2023. URL: <https://github.com/ARM-software/armnn> (visited on Sept. 15, 2026) (cit. on p. 42).
- [101] Intel Corporation. *OpenVINO 2022.1*. 2022. URL: <https://docs.openvino.ai/2022.1/index.html> (visited on Sept. 15, 2026) (cit. on p. 42).
- [102] N. Kokhlikyan, V. Miglani, M. Martin, et al. *Captum: A unified and generic model interpretability library for PyTorch*. arXiv preprint. 2020. DOI: 10.48550/arXiv.2009.07896 (cit. on p. 46).
- [103] M. D. Zeiler and R. Fergus. “Visualizing and Understanding Convolutional Networks”. In: *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014, pp. 818–833. DOI: 10.1007/978-3-319-10590-1_53 (cit. on p. 46).
- [104] R. R. Selvaraju, M. Cogswell, A. Das, et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: *International Journal of Computer Vision* 128.2 (2020), pp. 336–359. DOI: 10.1007/s11263-019-01228-7 (cit. on p. 46).
- [105] A. Mahendran and A. Vedaldi. “Salient Deconvolutional Networks”. In: *Computer Vision – ECCV 2016*. Cham: Springer International Publishing, 2016, pp. 120–135. DOI: 10.1007/978-3-319-46466-4_8 (cit. on p. 46).
- [106] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. *Striving for Simplicity: The All Convolutional Net*. International Conference on Learning Representations (ICLR 2015), Workshop Track Proceedings. 2015. DOI: 10.48550/arXiv.1412.6806. arXiv: 1412.6806 [cs.LG] (cit. on p. 46).
- [107] K. Simonyan, A. Vedaldi, and A. Zisserman. *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*. International Conference on Learning Representations (ICLR 2014), Workshop Track Proceedings. 2014. DOI: 10.48550/arXiv.1312.6034. arXiv: 1312.6034 [cs.CV] (cit. on p. 46).

- [108] W. Nie, Y. Zhang, and A. Patel. “A Theoretical Explanation for Perplexing Behaviors of Backpropagation-based Visualizations”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 3809–3818. URL: <https://proceedings.mlr.press/v80/nie18a.html> (visited on Sept. 15, 2026) (cit. on p. 46).
- [109] F. Lin, Y. Zhang, and J. Wang. “Recent advances in intra-hour solar forecasting: A review of ground-based sky image methods”. In: *International Journal of Forecasting* 39.1 (2023), pp. 244–265. DOI: 10.1016/j.ijforecast.2021.11.002 (cit. on p. 47).
- [110] J. Deng, W. Dong, R. Socher, et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on pp. 47, 86).
- [111] W. F. Holmgren, C. W. Hansen, and M. A. Mikofski. “pvlib python: a python package for modeling solar energy systems”. In: *Journal of Open Source Software* 3.29 (2018), p. 884. DOI: 10.21105/joss.00884 (cit. on p. 48).
- [112] W. Holmgren, K. Anderson, C. Hansen, et al. *pvlib/pvlib-python: v0.10.1*. Version v0.10.1. 2023. DOI: 10.5281/zenodo.8117210. (Visited on Sept. 15, 2026) (cit. on p. 48).
- [113] C. N. Long, J. M. Sabburg, J. Calbó, and D. Pagès. “Retrieving Cloud Characteristics from Ground-Based Daytime Color All-Sky Images”. In: *Journal of Atmospheric and Oceanic Technology* 23.5 (2006), pp. 633–652. DOI: 10.1175/JTECH1875.1 (cit. on p. 49).
- [114] Q. Li, W. Lu, and J. Yang. “A Hybrid Thresholding Algorithm for Cloud Detection on Ground-Based Color Images”. In: *Journal of Atmospheric and Oceanic Technology* 28.10 (2011), pp. 1286–1296. DOI: 10.1175/JTECH-D-11-00009.1 (cit. on p. 49).
- [115] T. Kroeger, R. Timofte, D. Dai, and L. Van Gool. “Fast optical flow using dense inverse search”. In: *Computer Vision – ECCV 2016*. Springer, 2016, pp. 471–488. DOI: 10.1007/978-3-319-46493-0_29 (cit. on pp. 49, 51).
- [116] G. Farnebäck. “Two-frame motion estimation based on polynomial expansion”. In: *Image Analysis*. Springer, 2003, pp. 363–370. DOI: 10.1007/3-540-45103-X_50 (cit. on pp. 49, 51).
- [117] S. Lambert-Lacroix and L. Zwald. “The adaptive BerHu penalty in robust regression”. In: *Journal of Nonparametric Statistics* 28.3 (2016), pp. 487–514. DOI: 10.1080/10485252.2016.1190359 (cit. on p. 49).
- [118] Z. Liu, H. Mao, C.-Y. Wu, et al. “A ConvNet for the 2020s”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 11966–11976. DOI: 10.1109/CVPR52688.2022.01167 (cit. on p. 50).
- [119] Y. Li, C.-Y. Wu, H. Fan, et al. “MViTv2: Improved Multiscale Vision Transformers for Classification and Detection”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 4794–4804. DOI: 10.1109/CVPR52688.2022.00476 (cit. on p. 50).

- [120] W. Yu, C. Si, P. Zhou, et al. “MetaFormer Baselines for Vision”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.2 (2024), pp. 896–912. DOI: 10.1109/TPAMI.2023.3329173 (cit. on p. 50).
- [121] H. Cai, J. Li, M. Hu, C. Gan, and S. Han. “EfficientViT: Lightweight Multi-Scale Attention for High-Resolution Dense Prediction”. In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023, pp. 17256–17267. DOI: 10.1109/ICCV51070.2023.01587 (cit. on p. 52).
- [122] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: <http://jmlr.org/papers/v12/pedregosa11a.html> (visited on Sept. 15, 2026) (cit. on p. 53).
- [123] H. Wen, Y. Du, X. Chen, et al. “Deep Learning Based Multistep Solar Forecasting for PV Ramp-Rate Control Using Sky Images”. In: *IEEE Transactions on Industrial Informatics* 17.2 (2021), pp. 1397–1406. DOI: 10.1109/TII.2020.2987916 (cit. on p. 53).
- [124] H. Zang, D. Chen, J. Liu, et al. “Improving ultra-short-term photovoltaic power forecasting using a novel sky-image-based framework considering spatial-temporal feature interaction”. In: *Energy* 293 (2024), p. 130538. DOI: 10.1016/j.energy.2024.130538 (cit. on p. 53).
- [125] T. M. Mercier, A. Sabet, and T. Rahman. “Vision transformer models to measure solar irradiance using sky images in temperate climates”. In: *Applied Energy* 362 (2024), p. 122967. DOI: 10.1016/j.apenergy.2024.122967 (cit. on p. 53).
- [126] A. L. Jonathan, D. Cai, C. C. Ukwuoma, et al. “A radiant shift: Attention-embedded CNNs for accurate solar irradiance forecasting and prediction from sky images”. In: *Renewable Energy* 234 (2024), p. 121133. DOI: 10.1016/j.renene.2024.121133 (cit. on p. 53).
- [127] M. Ansong, G. Huang, T. N. Nyang’onda, R. J. Musembi, and B. S. Richards. “Very short-term solar irradiance forecasting based on open-source low-cost sky imager and hybrid deep-learning techniques”. In: *Solar Energy* 294 (2025), p. 113516. DOI: 10.1016/j.solener.2025.113516 (cit. on p. 53).
- [128] N. Y. Hendriks, K. Barhmi, L. R. Visser, et al. “All sky imaging-based short-term solar irradiance forecasting with Long Short-Term Memory networks”. In: *Solar Energy* 272 (2024), p. 112463. DOI: 10.1016/j.solener.2024.112463 (cit. on p. 53).
- [129] G. Venitourakis, C. Vasilakis, A. Tsagkaropoulos, et al. “Neural Network-Based Solar Irradiance Forecast for Edge Computing Devices”. In: *Information* 14.11 (2023). DOI: 10.3390/info14110617 (cit. on pp. 53, 55, 56).
- [130] R. N. Hoffman, Z. Liu, J.-F. Louis, and C. Grassoti. “Distortion Representation of Forecast Errors”. In: *Monthly Weather Review* 123.9 (1995), pp. 2758–2770. DOI: 10.1175/1520-0493(1995)123<2758:DROFE>2.0.CO;2. URL: https://journals.ametsoc.org/view/journals/mwre/123/9/1520-0493_1995_123_2758_drofe_2_0_co_2.xml (cit. on p. 58).
- [131] E. D. Dunlop, F. Fabero, G. Friesen, et al. *Guidelines for PV power measurement in industry*. Tech. rep. European Commission, 2010. URL: <https://repository.supsi.ch/9717/> (visited on Sept. 15, 2026) (cit. on p. 65).

- [132] Q. Paletta, Y. Nie, Y.-M. Saint-Drenan, and B. Le Saux. “Improving cross-site generalisability of vision-based solar forecasting models with physics-informed transfer learning”. In: *Energy Conversion and Management* 309 (2024), p. 118398. DOI: 10.1016/j.enconman.2024.118398 (cit. on pp. 67, 68).
- [133] K. Roth, L. Pemula, J. Zepeda, et al. “Towards Total Recall in Industrial Anomaly Detection”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 14298–14308. DOI: 10.1109/CVPR52688.2022.01392 (cit. on p. 77).
- [134] R. Wightman. *PyTorch Image Models*. 2019. DOI: 10.5281/zenodo.4414861. (Visited on Sept. 15, 2026) (cit. on p. 85).
- [135] HAILO. *Hailo Dataflow Compiler*. Version 3.31.0. 2026. URL: <https://hailo.ai> (visited on Sept. 15, 2026) (cit. on p. 86).
- [136] A. Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. University of Toronto, 2009. URL: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf> (cit. on p. 86).
- [137] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. “Cats and dogs”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3498–3505. DOI: 10.1109/CVPR.2012.6248092 (cit. on p. 86).
- [138] HAILO. *HailoRT*. Version 4.23.0. 2026. URL: <https://hailo.ai> (visited on Sept. 15, 2026) (cit. on p. 86).
- [139] M. Kraft, B. Ptak, M. Piechocki, et al. “Visual Feedback System Supporting Robotic Manipulation of Hemp Plants”. In: *Journal of Natural Fibers* 22.1 (2025), p. 2454261. DOI: 10.1080/15440478.2025.2454261 (cit. on p. 118).
- [140] D. Pieczyński, B. Ptak, M. Kraft, M. Piechocki, and P. Aszkowski. “A fast, lightweight deep learning vision pipeline for autonomous UAV landing support with added robustness”. In: *Engineering Applications of Artificial Intelligence* 131 (2024), p. 107864. DOI: 10.1016/j.engappai.2024.107864 (cit. on p. 118).
- [141] B. Ptak, D. Pieczyński, M. Piechocki, and M. Kraft. “On-Board Crowd Counting and Density Estimation Using Low Altitude Unmanned Aerial Vehicles—Looking beyond Beating the Benchmark”. In: *Remote Sensing* 14.10 (2022), p. 2288. DOI: 10.3390/rs14102288 (cit. on p. 118).
- [142] M. Piechocki and A. Capotondi. “Benchmarking the Cost of Adaptation: A System-Level Analysis of Test-Time Adaptation on Edge Devices”. In: *Proceedings of the 24th Annual International Conference on Mobile Systems, Applications and Services Workshops*. Association for Computing Machinery, 2026, pp. 385–391. DOI: 10.1145/3812836.3814791 (cit. on p. 118).
- [143] W. Bałtrusiewicz, D. Belter, M. Bidziński, et al. “Robotics in Modern Agriculture Applications Implemented by Inwebit in Cooperation with Scientific and Research Institutions—Robotisation of Hemp Growth and Apple Harvesting”. In: *2024 13th International Workshop on Robot Motion and Control (RoMoCo)*. 2024, pp. 117–123. DOI: 10.1109/RoMoCo60539.2024.10604416 (cit. on p. 118).