MetaEdge: A Meta-Learning-Based Auto-Selective Tool for Hardware-Aware Anomaly Detection on Edge Devices

Nadia Rashid[1], Rashid Mehmood[2], Fahad Alqurashi[3], Turki Alghamdi[4]
Department of Computer Science-FCIT, King Abdulaziz University, Jeddah 21589, Saudi Arabia[1, 3]
College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj 16278, Saudi Arabia[1]
Faculty of Computer and Information Systems, Islamic University of Madinah, Madinah 42351, Saudi Arabia[2, 4]

Abstract—**The deployment of anomaly detection systems across heterogeneous edge computing environments faces significant challenges due to varying computational constraints and resource limitations. Existing approaches typically employ static model selection strategies that fail to adapt to diverse hardware capabilities, resulting in suboptimal detection performance and inefficient resource utilization. To address this, we propose MetaEdge, a novel hardware-aware framework that intelligently selects and deploys anomaly detection models based on specific device characteristics and hardware constraints. The MetaEdge framework introduces a systematic methodology that leverages meta-learning in the first stage to train a machine learning model to predict the top-k anomaly detectors by considering dataset characteristics. These candidates are then put through hardware-aware optimization that incorporates the hardware constraints of edge devices to ensure deployment feasibility. The framework evaluates 11 candidate anomaly detection algorithms spanning traditional machine learning and deep learning methods across four representative computing architectures ranging from ultra-constrained edge devices to GPU-accelerated cloud instances. Model conversion through ONNX standardization enables cross-platform deployment while maintaining detection capabilities. Experimental evaluation demonstrates the framework's effectiveness in achieving superior anomaly detection performance across diverse hardware configurations. The hardware-aware stage successfully identifies optimal model-hardware pairings, with the deployed models achieving up to 96.6% accuracy and 90.4% precision on edge devices. The framework demonstrates high accuracy in model selection decisions, with confidence scores providing meaningful hardware compatibility assessments that guide deployment. MetaEdge introduces a novel paradigm for hardware-aware anomaly detection in edge computing, demonstrating that meta-learning–driven model selection can deliver superior detection performance while adhering to stringent hardware constraints. By integrating automatic model selection with hardware-aware optimization, the proposed approach enables anomaly detection systems to intelligently adapt to diverse computing environments and maximize performance under resource constraints.**

Keywords—Anomaly detection; edge computing; hardware-aware optimization; machine learning; meta-learning; model selection; ONNX

I. Introduction

The rapid expansion of the Internet of Things (IoT) has produced an extraordinary volume of data at the network edge. Billions of heterogeneous devices ranging from industrial sensors and autonomous vehicles to wearable health monitors generate continuous, high-velocity data streams. Real-time analysis of these data streams is crucial for extracting value and supporting intelligent applications. To meet this latency and bandwidth demands while strengthening data privacy, computation is increasingly migrating from centralized cloud infrastructures to distributed edge nodes. This architectural shift is necessary for applications that demand low latency, high bandwidth, and stronger data privacy; however, it also poses substantial challenges, especially for deploying advanced machine learning models on resource-constrained edge devices. Addressing this capability–constraint trade-off has become a central challenge for next-generation IoT systems [1], [2].

Anomaly detection, which involves identifying rare events or observations that deviate significantly from normal behavior, is one of the most critical tasks performed at the edge. Its importance spans domains: it supports predictive maintenance and fault avoidance in industrial systems, enables real-time patient monitoring and alerts in healthcare, and identifies intrusion and malware detection in cybersecurity [3]. However, deploying anomaly detection models on edge devices remains highly challenging. These devices operate under stringent constraints on computation, memory, and energy [4]. Offloading raw data to the cloud is often impractical given strict real-time requirements, privacy considerations, and communication costs [1]. Consequently, performing anomaly detection directly on the device, i.e., edge inference, has become increasingly necessary. Selecting an effective anomaly detection model further complicates edge deployment. Model performance depends strongly on data characteristics and anomaly types, and no single model is universally optimal [5]. Consequently, manual model selection becomes time-consuming and expertise-intensive, rendering it impractical for large-scale, heterogeneous edge environments. These limitations have motivated the development of automatic model selection tools that automate the end-to-end process from model choice to tuning. Within this paradigm, meta-learning approaches are particularly promising. Meta-learning leverages performance evidence from diverse prior tasks to recommend effective models for new, unseen tasks. It improves the performance on similar tasks and reduces costly, time-consuming retraining [6].

Even when a meta-learning system recommends an optimal model, it may still be infeasible if it exceeds the hardware capabilities of the target edge device. A performance-optimized deep learning model, for instance, may demand more memory or computational resources than the device can supply [4]. This necessitates hardware-aware model selection in which device constraints are a primary consideration. The selected model must strike an appropriate balance between predictive performance and computational efficiency. This imperative has driven extensive work on model optimization techniques, such as quantization, pruning, and hardware-aware architecture design, as well as deployment-friendly runtimes such as TensorFlow Lite, to fit powerful models onto constrained platforms [7].

This study presents MetaEdge, a framework for automatic, hardware-aware selection of anomaly detection models for edge inference. This framework optimizes the trade-off between model accuracy and resource efficiency in a fully automated manner. First, a meta-learning-based model selection engine utilizes the meta-features of a given streaming dataset to generate a ranked list of promising anomaly detection models from a predefined candidate pool. Second, a hardware-aware selection module filters the ranked list using the specified hardware constraints of the target edge device, such as available memory, processor capabilities, and latency requirements, returning the best model that maximizes performance while remaining within those operational limits. Where applicable, the selected model is compiled into a lightweight deployment artifact (e.g., TFLite and ONNX) to ensure efficient on-device inference. Together, this end-to-end approach yields models that are not only accurate but also practical for real-world edge deployment.

The novelty of MetaEdge lies in three key aspects that distinguish it from prior work. First, while meta-learning approaches (e.g., AMLBID [6], MetaOOD [12]) have been applied to model selection and hardware-aware optimization techniques (e.g., LightESD [16], EdgeML [11]), they have been developed for edge deployment. MetaEdge is the first to systematically integrate both paradigms into a unified end-to-end framework specifically designed for anomaly detection on heterogeneous edge devices. Second, MetaEdge introduces a comprehensive synthetic hardware-learning dataset (254 configurations, 2,794 training samples) that explicitly captures the relationships between hardware constraints (memory, latency, energy) and model deployment characteristics. This dataset enables the hardware-aware module to learn realistic constraint-performance trade-offs, a capability absent in prior frameworks that rely on heuristic or rule-based hardware filtering. Third, MetaEdge provides empirical validation across real physical edge devices (Raspberry Pi 5, Orange Pi Zero 2W) combined with simulated cloud-edge profiles (AWS EC2 t2.micro, g4dn.xlarge), demonstrating practical deployment feasibility. This real-world validation, coupled with detailed per-device performance analysis, goes beyond the simulated or cloud-based evaluations common in prior work. Together, these contributions establish MetaEdge as a comprehensive, data-driven, and practically validated framework that advances the state-of-the-art in hardware-aware anomaly detection for edge computing.

The main contributions of this work are summarized as follows:

- Introduce MetaEdge, a framework that couples meta-learning-based model selection with hardware-aware selection for edge anomaly detection, narrowing the gap between theoretical performance and deployment on resource-constrained devices. The meta-learning module recommends the top-k models based on dataset characteristics, and the hardware-aware module evaluates these k candidates against target hardware specifications to select the model that best balances performance and deployment feasibility.
- Construct a hardware-learning dataset comprising 254 synthetic hardware configurations and comprehensive profiling of 11 candidate models, yielding 2,794 training samples. This dataset captures relationships between hardware constraints and model deployment characteristics across diverse scenarios.
- Design an efficient hardware-aware module that translates performance-based recommendations from the meta-learning module into deployment-ready selections under real-world hardware constraints. The module turns the hardware-learning dataset into a practical decision-making tool and outputs a probability distribution over all models (confidence scores), where each probability reflects the likelihood that a given model is the best choice for the specified hardware.
- Provide an empirical evaluation across a variety of practical edges, including two real devices (Raspberry Pi 5 and Orange Pi Zero 2W) using Amazon Web Services (AWS) Greengrass V2 and two simulated devices (g4dn.xlarge and a t2.micro) on AWS EC2. We also experiment with different streaming buffer sizes under various data-processing settings, giving us a well-rounded, deployment-ready assessment.

MetaEdge is evaluated along four dimensions: meta-learning-based model selection, hardware-aware model selection, model conversion effectiveness, and deployment feasibility. We first examined how available data volume affects model selection by varying the buffer size (500 vs. 1,000 samples). For each condition, the meta-learning module ranked candidate anomaly detectors and returned the top k models. Providing more buffered data makes the meta-learner both more confident and more accurate in its recommendations. Second, we tested the model conversion effectiveness by converting the models to lightweight deployment versions and comparing them with its original counterparts on two metrics: model file size and accuracy. The accuracy losses were minimal with only 0–3.8% degradation, and the size reductions were substantial of 55-70% across all models. Third, we validated the hardware-aware module by executing it in live deployments across four edge targets. Two were physical devices, a Raspberry Pi 5 (higher-end) and an Orange Pi Zero 2W (resource-constrained) and two were emulated edge profiles on AWS EC2: a t2.micro to represent a low-resource CPU environment and a g4dn.xlarge to represent a GPU-enabled, Jetson-like setting. For each device, we assess both the real-time feasibility and the anomaly detection performance of the selected model. This evaluation confirmed that MetaEdge consistently identified optimal model–hardware pairings, with edge implementations achieving up to 96.6% accuracy and 90.4% precision.

The remainder of this study is organized as follows: Section II reviews related work and identifies the research gap. Section III details the architecture and methodology of our proposed MetaEdge framework. Section IV presents the experimental setup and a comprehensive evaluation. Finally, Section V concludes the study and discusses future research directions.

II. Related Work

This section provides a structured survey of the literature relevant to our automated, hardware-aware model selection framework for edge anomaly detection. We organize the discussion around three strands that collectively motivate our approach. First, we examine anomaly detection at the edge, with an emphasis on the distinctive constraints of resource-constrained devices and real-time processing, to establish the problem setting and the computational limits that necessitate intelligent model selection. Second, we review automated model selection and meta-learning approaches, evaluating how current frameworks leverage historical performance and meta-features to streamline decision-making while exposing gaps in device-level considerations. Third, we investigate hardware-aware optimization techniques for resource-constrained platforms, clarifying how current approaches address resource constraints and deployment challenges on edge devices. This analysis highlights the importance of incorporating device-level constraints directly into the model selection process. It also reveals critical gaps in existing work that fail to address the end-to-end path from model selection to hardware-constrained deployment, which motivates the comprehensive framework introduced in this study.

A. Lightweight Anomaly Detection for Edge Devices

Research on edge anomaly detection has gained significant attention with the proliferation of IoT devices and the attendant need for real-time decisions. Chatterjee and Ahmed [8] present an extensive survey of IoT anomaly detection methods, highlighting key challenges including limited computational resources, evolving normal behavior patterns, and the scarcity of labeled data. Their examination of 64 recent studies reveals a notable deficit of approaches for multi-sensor integration and concept drift management on constrained devices. Complementing this perspective, Jadhav and Kulkarni [4] examine anomaly detection within edge-computing networks, with particular attention to deep learning methods. Their survey underscores that anomalies at the edge pose substantial risks to enterprise networks and that monitoring and identifying abnormal behavior grow increasingly difficult as interconnections increase. While deep models appear promising, the authors underscore the difficulty of fitting such models within the memory and latency budgets of edge devices. Recent practical implementations have demonstrated both the strengths and the limitations of edge anomaly detection. Reis et al. [10] combine Isolation Forests with LSTM autoencoders within an edge AI framework for smart-home applications, achieving respectable accuracy but only with careful optimization for resource-constrained devices. Similarly, Patrikar and Parate [9] demonstrate video-surveillance anomaly detection at the edge, reporting substantial latency reductions while highlighting the inherent trade-off between model complexity and inference speed. Das et al. [12] propose LightESD, a fully automated, lightweight anomaly-detection method based on statistical learning. LightESD runs on-device without transferring data between the edge and the server and is designed for low latency, memory usage, and energy consumption, making it suitable for low-end edge hardware. The framework maintains competitive detection accuracy while consuming extremely low resources, enabling deployment on low-end edge devices. However, LightESD does not address hardware-aware model selection across heterogeneous devices and lacks explicit mechanisms for handling streaming concept drift in long-running deployments. It also focuses exclusively on statistical methods and does not incorporate meta-learning approaches for intelligent model selection, limiting its adaptability across diverse data characteristics. Collectively, these works frame the problem and underscore that effective solutions must balance detection quality with stringent resource constraints. The next subsection considers automated model selection and meta-learning as means to streamline selection.

B. Automated Model Selection for Edge Computing

The challenge of selecting appropriate machine learning models has motivated the development of automated model selection frameworks. Ying et al. [10] propose an automated model-selection framework for time-series anomaly detection that identifies suitable models and hyperparameters via an extensible selection layer and customized tuning. However, the design targets cloud-based deployments and leaves device constraints out of scope. In contrast, EdgeML [11] is an AutoML framework purpose-built for real-time deep-learning applications on edge devices. It addresses practical obstacles such as limited CPU resources and energy budgets and automates model selection and hyperparameter tuning to enable efficient on-device deployment and real-time inference without reliance on cloud resources. It additionally adapts to runtime conditions such as communication bandwidth fluctuations and varying computational loads. However, EdgeML focuses exclusively on deep neural network optimization and does not address the broader challenge of selecting appropriate anomaly detection algorithms based on data characteristics. Additionally, it lacks meta-learning capabilities for algorithm recommendation and is limited to neural network architectures rather than providing a comprehensive model selection framework. Meta-learning has emerged as an effective approach to automated model selection. Garouani et al. [6] introduce AMLBID, a meta-learning tool for industrial big data that achieves near O(1) complexity for model selection. The framework maintains a meta-knowledge base of algorithm performance across datasets and exploits meta-features to recommend optimal pipelines. Although effective in conventional computing settings, it does not address the distinctive constraints of edge deployments. Most recently, MetaOOD by Qin et al. [12] advances meta-learning for anomaly detection by introducing the first zero-shot, unsupervised framework for selecting out-of-distribution (OOD) detectors. Using language-model embeddings to represent datasets and models, it reports superior performance across 24 dataset pairs and 11 detectors. Nevertheless, the framework operates under effectively unconstrained compute budgets and omits hardware-aware deployment considerations. These limitations motivate a shift toward hardware-aware machine learning, which explicitly incorporates device-level constraints into model design and deployment. The next subsection reviews this body of literature.

C. Hardware-Adaptive Model Optimization

The constraints of edge devices make hardware-aware optimization indispensable for the deployment of machine learning models. Shuvo et al. [2] present techniques for accelerating deep-learning inference on edge devices, outlining four major directions: novel architectures, optimization of existing methods, algorithm–hardware co-design, and efficient accelerator development. Their findings highlight the necessity of simultaneous hardware- and software-level optimization. Murshed et al. [1] review machine learning deployment at the network edge, outlining the core trade-offs between cloud processing and edge inference. Whereas the cloud offers vast compute capacity, edge deployment directly addresses latency, communication overhead, and privacy. The survey underscores the need for compression techniques, specialized tooling, and hardware-aware frameworks to make edge deployment practical. Researchers have explored multiple approaches to address the challenges of rapidly deploying machine-learning models on edge devices. One effective strategy is to design lightweight architectures specifically for edge deployment. Such architectures employ techniques such as depthwise-separable convolutions, low-rank factorization, and network pruning to improve efficiency without sacrificing accuracy [13]. In this context, MobileNet and ShuffleNet have emerged as popular choices for on-device image classification due to their compact designs and favorable accuracy–latency trade-offs on resource-limited hardware [14]. However, these architectures are mostly vision-centric and often must be retrained for each task and dataset; moreover, without optimized kernels, their speedups are hard to realize, especially for non-vision workloads or devices lacking appropriate operators. Another area of research is model compression for reducing the size and computation of neural networks. Techniques such as quantization and knowledge distillation have shown strong potential to shrink models while preserving accuracy. Pruning and structured sparsity further remove redundant parameters and, when supported by the runtime or hardware, can deliver real latency and energy savings, making them attractive for edge deployment [13]. However, compression can make models less robust when the data changes, hurt calibration, and miss rare events. The gains also depend on hardware/runtime support for low-precision or sparse operations (ops), and distillation needs a strong teacher, complicating use in streaming edge anomaly detection. Hardware-aware neural architecture search has become a promising strategy for tailoring models to specific deployment targets. Lee et al. [15] introduce HELP, a Hardware-adaptive Efficient Latency Predictor formulated via meta-learning to estimate device-specific performance. While compelling for neural architecture search, it addresses architecture design rather than model selection for anomaly detection and does not engage streaming constraints. Collectively, these studies underscore the necessity of embedding device constraints across the entire pipeline. The next subsection offers a research gap analysis that integrates these insights and motivates our proposed framework.

D. Research GAP

Table I presents a comparative analysis of existing approaches across deployment-relevant dimensions for edge-oriented anomaly detection model selection. The evaluation criteria include: Meta-Learning (whether the approach uses meta-learning techniques to automatically select models), Hardware-Aware (explicit consideration of computational and memory constraints during model selection), Edge-Specific (design tailored for edge computing environments with their unique limitations), Anomaly Detection (focus on anomaly detection tasks rather than general machine learning), Streaming Data (capability to handle real-time data streams), and Real Device Validation (empirical evaluation on real physical edge devices rather than simulations). As shown in Table I, existing work reveals several gaps for edge-based anomaly detection. Although automated model selection for anomaly detection exists [10] and meta-learning has been explored for general machine learning [6], [12], yet none of them combine meta-learning with hardware-aware selection tailored to edge-specific constraints.

Recent edge-specific frameworks such as LightESD [16] and EdgeML [11] demonstrate significant progress in addressing hardware constraints and edge deployment challenges, but both lack meta-learning capabilities for intelligent model selection. LightESD focuses exclusively on statistical methods without the ability to adapt to diverse data characteristics, while EdgeML is limited to deep neural network optimization and does not address the broader spectrum of anomaly detection algorithms. Hardware-aware approaches either concentrate on neural architecture search (e.g., HELP [15]) or offer ad hoc solutions for specific applications [9], [17], leaving the field without a rigorous, constraint-driven model selection strategy that explicitly optimizes under device-level constraints. Moreover, commonly used meta-learning paradigms implicitly assume ample computation and fail to address the real-time, streaming nature of edge workloads and their stringent resource budgets. Evaluation practices likewise rely on simulated environments or cloud-based deployments, with limited validation on real devices, despite the well-known divergence between reported metrics and performance on constrained hardware. These limitations motivate a unified framework that couples meta-learning-based recommendation with hardware-aware model selection for streaming anomaly detection at the edge. This study proposed the MetaEdge framework that spans the entire pipeline, from characterizing streaming data to selecting a deployment-ready model, while explicitly accounting for the practical constraints of real-world edge deployments.

TABLE I. COMPARATIVE SUMMARY OF RELATED APPROACHES ACROSS DEPLOYMENT-RELEVANT DIMENSIONS

Approach	Meta-Learning	Hardware-Aware	Edge-Specific	Anomaly Detection	Streaming Data	Real-Device Validation
Ying et al. [10]	✗	✗	✗	✓	✓	✗
AMLBID [6]	✓	✗	✗	✗	✗	✗
MetaOOD [12]	✓	✗	✗	✓	✗	✗
HELP [15]	✓	✓	✓	✗	✗	✗
Edge AI frameworks [9], [17]	✗	✓	✓	✓	✓	✓
LightESD [16]	✗	✗	✓	✓	✓	✓
EdgeML [11]	✗	✓	✓	✗	✓	✓
This work	✓	✓	✓	✓	✓	✓

The novelty of MetaEdge lies in how it bridges the gap between data-driven model recommendation and hardware-aware deployment. A pure meta-learning approach (e.g., AMLBID [6]) would recommend models based solely on data characteristics, ignoring hardware constraints and potentially selecting models that are not deployable on the target device. Conversely, a pure hardware-aware selection approach would select the most efficient model that fits within the hardware budget, but such a model may exhibit poor detection performance for the given data. Existing tools, such as EdgeML [11], focus on hardware-aware optimization for deep learning models yet lack a meta-learning component for recommending the most suitable model class based on data characteristics. LightESD [16] provides lightweight statistical methods but does not incorporate meta-learning or hardware-aware model selection across heterogeneous devices. MetaEdge, in contrast, integrates these two dimensions, using meta-learning to identify a shortlist of high-performing models for the data and then applying a hardware-aware module to select the model that best satisfies device constraints while maintaining strong detection performance. This two-stage approach provides a more holistic solution than existing methods by balancing data-driven accuracy and hardware-driven feasibility.

III. METHODOLOGY AND DESIGN

This section presents MetaEdge, an automated, hardware-aware model selection framework for anomaly detection at the edge. MetaEdge couples meta-learning–driven performance prediction with device-constrained selection to identify a single model that delivers high detection performance while meeting the computational, memory, and latency budgets of the target device. To maintain simplicity and reproducibility, the process is organized into a two-stage pipeline: meta-learning-based selection followed by hardware-aware selection. We first provide a comprehensive overview of the MetaEdge architecture, then present detailed descriptions of each component.

A. Framework Overview

Fig. 1 illustrates a two-stage pipeline for the MetaEdge framework. In the offline training (cloud) stage, training datasets and candidate ML/DL models are applied to meta-feature extraction and model evaluation to build a meta-knowledge dataset, which is transformed into a meta-learning dataset for training a meta-learning module (based on ASAD). In parallel, model profiling combined with hardware datasets produces a hardware-learning dataset, which is used to train a hardware-aware module. These two trained modules constitute the core of the MetaEdge engine. In the online selection (edge) stage, an edge device provides streaming data and hardware specifications; meta-features are extracted and fed into the MetaEdge, which selects and deploys the most suitable, hardware-optimized model for the device. Deployment outcomes and anomaly-detection signals form a feedback loop that enables the engine to adaptively select models on-device, guided by both data characteristics and hardware constraints.

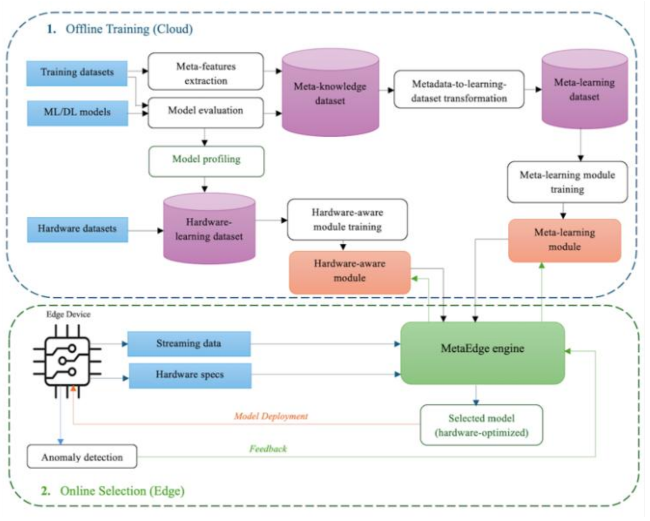


Fig. 1. The architecture of the MetaEdge framework.

MetaEdge integrates its meta-learning and hardware-aware components to select and deploy a hardware-optimized model on the device, demonstrating how historical performance data and hardware constraints can be systematically combined to deliver intelligent, deployment-ready model selection for edge environments, bridging the gap between theoretical model effectiveness and real-world applicability.

B. Meta-Learning Selection Module

The meta-learning engine forms the first stage of our framework, building upon our previous work, ASAD (Auto-Selective Anomaly Detection) [18]. We follow the same meta-learning logic and methodology established in ASAD, which leverages historical performance data across diverse datasets to recommend the most promising anomaly detection models for new, unseen data. The meta-learning stage operates offline to construct a large meta-knowledge dataset for model recommendation. A diverse benchmark of labeled anomaly detection datasets (139 datasets derived from 60 bases across 11 domains) is assembled, and a broad set of candidate models is evaluated under a unified protocol and seven standard effectiveness metrics. In parallel, an extensive bank of 300 dataset-level meta-features is computed spanning six families: simple, statistical, information-theoretic, model-based, landmarking, and data-complexity to capture dataset size/shape, distributional behavior, and structural dependencies. These meta-features reliably characterize new datasets and guide accurate model selection. The result is a models-by-datasets performance matrix paired with per-dataset meta-feature vectors. This dataset is then transformed into a meta-learning dataset in which each instance contains dataset meta-features and a learning target that encodes the identity of the best-performing model. A supervised meta-learner is trained to map meta-features to predict performance using standard train-validation splits to avoid leakage across datasets. At inference time, the same meta-features are extracted from the incoming unseen dataset, and the meta-learner returns a ranked list of anomaly detectors, enabling fast, trial-free recommendation for new datasets.

In this edge-focused framework, we restrict the candidate model pool to eleven models (AE, MLP, GRU, LSTM, DeepSVDD, GAN, LOF, OCSVM, KMeans, GMM, and PCA) instead of the 80 ML/DL models used in the original ASAD approach. This restriction is specifically designed to prioritize models that are suitable for edge deployment while maintaining the proven effectiveness of the ASAD meta-learning methodology. To ensure deployability, each candidate must be convertible into a tiny, edge-ready artifact via TensorFlow Lite (TFLite) or Open Neural Network Exchange (ONNX) runtime, with support for quantization and lightweight inference. This toolchain requirement is precisely why we do not reuse the broader PyOD-based model list from ASAD: many of those implementations lack reliable TFLite/ONNX export paths and therefore do not meet strict latency/memory targets. Instead, we focus on a curated set that preserves coverage across reconstruction, sequence modeling, clustering, margin-based one-class classification, and local-density detection, chosen for their practical convertibility, stable runtime support, and compatibility with post-training optimizations, thereby preserving methodological rigor while guaranteeing real-world deployability. The current system is positioned as a proof-of-concept, demonstrating end-to-end feasibility under these toolchain constraints; the candidate pool can be expanded as TensorFlow Lite (TFLite) or ONNX support improves without altering the selection logic.

C. Hardware-Aware Selection Module

The hardware-aware selection module represents the second stage of our framework and constitutes our primary novel contribution. This module addresses the critical challenge of translating performance-based model recommendations from the meta-learning module into deployment-ready selections that satisfy real-world hardware constraints. The module operates through a systematic four-step process: synthetic dataset generation, model profiling, hardware learning dataset construction, and hardware-aware model training.

1) Hardware dataset generation: The foundation of hardware-aware model selection begins with the creation of a comprehensive synthetic dataset that represents the diversity of edge computing environments. This step is crucial because it establishes the training basis for learning hardware-performance relationships across different device configurations. We generated 254 synthetic samples, each representing a unique hardware configuration encountered in edge scenarios. The sample size was chosen to provide sufficient coverage of the space of deployment scenarios while maintaining computational tractability for training. Each scenario represents a unique combination of hardware constraints and deployment requirements and is characterized by seven critical features that directly impact model deployment feasibility. The seven features used in the hardware dataset are presented in Table II.

TABLE II. THE SEVEN FEATURES USED IN THE HARDWARE DATASET

Hardware Constraint Features		
Feature	**Definition**	**Importance**
Max_Ram	Maximum available RAM in megabytes	Represent device memory capacity limits
Max_Inference	Maximum acceptable inference latency in milliseconds	Capture real-time performance requirements
Max_Size	Maximum allowable model size in megabytes	Reflect storage and bandwidth constraints
Max_Energy	Maximum energy consumption per inference in millijoules	Address battery-powered device limitations
Deployment Requirement Features		
Feature	**Definition**	**Importance**
Requires_Tflite	A binary indicator (1/0) for TensorFlow Lite format	Essential for mobile deployment optimization
Requires_Onnx	A binary indicator (1/0) for ONNX format	Support cross-platform compatibility
GPU_Available	A binary indicator (1/0) for GPU availability	Enabling hardware-specific optimization

The deployment scenarios were generated using an equal distribution approach with balanced representation across distinct deployment contexts. The process reflects realistic combinations observed in commercial edge devices, ensuring comprehensive coverage of the deployment space. These synthetic hardware configurations serve as the foundation for constructing diverse deployment scenarios. Each configuration represents a distinct set of device capabilities and constraints that will later be used to determine optimal model selection based on constraint satisfaction and deployment requirements.

2) Comprehensive model profiling: Model profiling is the bridge between algorithmic performance and deployment reality. It extracts deployment-specific characteristics that are essential for hardware-aware decision making but are often overlooked by traditional model selection. During training, all 11 candidate models are comprehensively profiled so that the hardware-aware selector learns a complete mapping from hardware constraints to deployment feasibility across the entire model space. This comprehensive approach is essential because the selector must understand the deployment characteristics of every possible model choice to make informed, hardware-aware decisions.

Profiling spans the memory and storage footprint, runtime efficiency, and format compatibility. On the memory side, runtime RAM during inference ranges from 0.02 MB for PCA to 0.98 MB for LSTM, while on-disk model size ranges from 0.001 MB for KMeans and PCA to 0.113 MB for LSTM. For hardware-accelerated inference, GPU memory requirements vary from 0 MB for traditional ML models to 90 MB for GANs. Efficiency metrics include single-inference latency from 0.68 ms (PCA) to 527.63 ms (LSTM) and energy on MCUs, from 0.04 mJ (PCA) to 29.02 mJ (LSTM). Format compatibility is captured through TFLite and ONNX: deep learning models support TFLite conversion, typically yielding a 10–15% size reduction relative to the original model, whereas traditional ML models are exported to ONNX for cross-platform deployment; corresponding TFLite and ONNX sizes are logged for each model.

The profiling reveals distinct deployment characteristics across model categories. Deep learning models, AutoEncoder, MLP, GRU, LSTM, DeepSVDD, and GAN, support TFLite optimization for mobile targets, benefit from GPU acceleration (30–90 MB GPU RAM), and generally require higher resource usage (0.77–0.98 MB RAM, 11.54–29.02 mJ per inference) with moderate-to-high latency (209.79–527.63 ms). Traditional machine-learning models, LOF, OCSVM, KMeans, GMM, and PCA, export cleanly to ONNX, require minimal resources (0.02–0.07 MB RAM, 0.04–0.17 mJ per inference), deliver ultra-low latency (≈0.68–3.11 ms), and do not require GPU acceleration. For computational efficiency, the strategy differs between training and inference. Training profiles all 11 models to cover the full deployment characteristic space. At inference time, only the top K recommended models from the meta-learning stage are profiled, which reduces profiling overhead by around 64% on average when k=4 (from 11 to 4 models) while preserving selection quality within the constrained candidate set.

3) Hardware-learning dataset construction: The hardware-learning dataset construction represents the critical integration phase that combines the synthetic hardware configurations from Step 1 (Hardware Dataset Generation) with the comprehensive model profiling results from Step 2 (Comprehensive Model Profiling). This step produces the final training dataset with ground truth labels that are determined through a hardware-focused scoring algorithm. It captures the complex relationships between hardware constraints and model deployment characteristics across diverse deployment scenarios.

The hardware-learning dataset construction process expands the 254 synthetic hardware configurations from Step 1 into a comprehensive training dataset. Using an equal distribution approach, each of the 254 hardware configurations is combined with all 11 candidate models, resulting in 2,794 total training samples (254 configurations × 11 models). This balanced distribution ensures that the meta-selector receives equal learning exposure to each model's deployment characteristics, preventing bias toward any algorithm. The integration with model profiling data enables the selection of the optimal model for each scenario through systematic constraint evaluation.

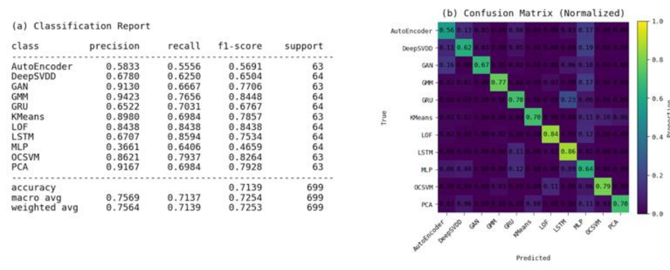
$$FinalScore = Eff_{Score} + Format_{Bonus} + Device_{Bonus} \quad (1)$$

The ground truth label assignment employs a hardware-aware scoring algorithm that determines the best-fitting model for each deployment scenario. Each candidate model is assigned a composite suitability score computed as the sum of an efficiency-based score (Eff_{Score}) and deployment-specific bonuses ($ormat_{Bonus}$ and $Devic\,e_{Bonus}$). The final score is defined as in Eq. (1). The efficiency score is calculated as a weighted linear combination of normalized resource metrics, given by Eq. (2):

$$Eff_{Score} = 0.3 \cdot RAM_{eff} + 0.3 \cdot Speed_{eff} + 0.2 \cdot Size_{eff} + 0.2 \cdot Energy_{eff} \quad (2)$$

The hardware-aware scoring algorithm prioritizes multiple factors, including memory efficiency, speed, format compatibility, and device-specific constraints, in a weighted evaluation. It first applies hard constraints to filter candidates: any model that violates critical limits on Max_Ram, Max_Inference, Max_Size, or Max_Energy is eliminated; models incompatible with required formats (Requires_Tflite, Requires_Onnx) are filtered out; and GPU needs must match device capabilities (Gpu_Available). Among the remaining models, those with lower RAM requirements receive higher scores in memory-constrained scenarios, while faster inference is favored for latency-sensitive deployments. Support for the required format (TFLite or ONNX) earns a compatibility bonus, and GPU availability together with energy limits further adjust each model's overall suitability score. For each of the 2,794 scenarios, the algorithm evaluates all 11 candidate models against the scenario's specific constraints. After constraint filtering and priority-weighted scoring, the model with the highest composite score becomes the ground truth label (BestModel) for that scenario. This approach ensures that the hardware-aware selector learns realistic deployment decisions that reflect both technical constraints and deployment priorities.

4) Hardware-aware model training and optimization: The final step involves training a classifier to predict the optimal model given the integrated hardware and deployment features. This step transforms the comprehensive dataset into a practical decision-making tool for hardware-aware model selection. A Random Forest was selected as the hardware-aware selection algorithm due to its ability to handle mixed feature types, robustness to feature scaling differences, and interpretability through feature importance analysis [19]. The training process starts with data preprocessing, including standardization and label encoding to ensure consistent feature scaling and proper handling of categorical variables. An 80/20 stratified train-test split maintains balanced class representation across all 11 anomaly detection models. The training step produces a classifier that predicts the optimal model selection among all 11 candidate anomaly detection algorithms based on hardware constraint features. It also outputs a probability distribution (confidence scores) over all models, where each probability indicates the likelihood that a particular model is the best choice for the specified hardware.



```
(a) Classification Report

class            precision     recall   f1-score    support
-----------------------------------------------------------
AutoEncoder         0.5833     0.5556     0.5691         63
DeepSVDD            0.6780     0.6250     0.6504         64
GAN                 0.9130     0.6667     0.7706         63
GMM                 0.9423     0.7656     0.8448         64
GRU                 0.6522     0.7031     0.6767         64
KMeans              0.8980     0.6984     0.7857         63
LOF                 0.8438     0.8438     0.8438         64
LSTM                0.6707     0.8594     0.7534         64
MLP                 0.3661     0.6406     0.4659         64
OCSVM               0.8621     0.7937     0.8264         63
PCA                 0.9167     0.6984     0.7928         63
-----------------------------------------------------------
accuracy                                  0.7139        699
macro avg           0.7569     0.7137     0.7254        699
weighted avg        0.7564     0.7139     0.7253        699
```

Fig. 2. Performance of the hardware-aware classifier: (a) Classification report; (b) Normalized confusion matrix.

Fig. 2 shows strong, class-consistent performance across 11 categories. The multi-class model achieved 71.4% overall accuracy (699 samples), with macro-averaged precision 0.7569, recall 0.7137, and F1-score 0.7254, indicating balanced performance across classes. The row-normalized confusion matrix shows a clear, dominant diagonal where most classes (7 out of 11) achieve ≥ 0.70 recall, with particularly strong performance for LSTM (0.86), LOF (0.84), OCSVM (0.79), and GMM (0.77). Off-diagonal entries are low and diffuse, indicating minimal systematic confusion and confirming the model's ability to reliably discriminate among the competing approaches. These results represent a significant improvement over random selection and validate the effectiveness of constraint-driven feature-engineering and hardware-focused scoring.

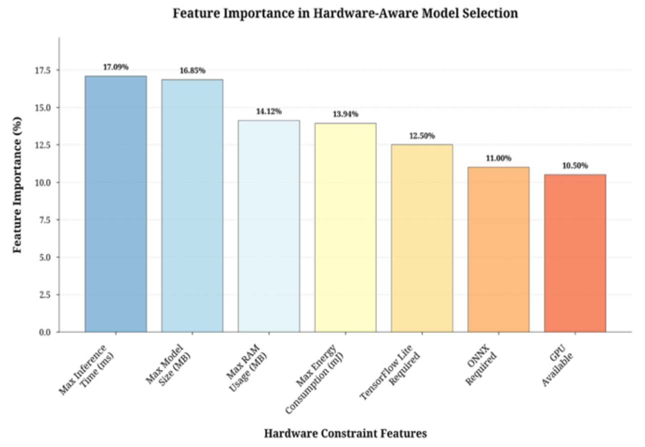


Fig. 3. Feature importance analysis of the hardware-aware model selection.

The Random Forest feature importance analysis reveals the relative contribution of each input feature to the selection decision, as illustrated in Fig. 3. The feature-importance analysis indicates that deployment constraints tied to latency and footprint dominate model selection: Max_Inference contributes the most at 17.09%, followed by Max_Size at 16.85%. Resource limits are next with Max_RAM (14.12%) and Max_Energy (13.94%), highlighting the importance of running efficiently on devices with limited memory and power. Platform compatibility matters but is secondary, with Requires_Tflite (12.50%) and Requires_Onnx (11.00%). GPU Availability has the lowest but still meaningful impact (10.50%), suggesting decisions are driven more by edge-device constraints than by access to accelerators.

D. MetaEdge Engine Construction

The MetaEdge engine is constructed by seamless integration of the hardware-aware selection module and the meta-learning module through a two-stage decision process. During inference, the meta-learning module first generates top-k model recommendations based on dataset characteristics. The hardware-aware module then evaluates these k candidates against the target hardware specifications to select the best candidate model that balances performance potential with deployment feasibility. A key design decision in MetaEdge is this two-stage architecture: by first filtering with meta-learning and then applying hardware-aware optimization, the system constrains the hardware-aware module to the top-K models while gaining important practical advantages, including computational efficiency by reducing the candidate pool, interpretability through a clear separation of data-driven and hardware-driven decisions, and proven effectiveness across diverse deployment scenarios. This design prioritizes data-driven model selection as the primary driver, using hardware constraints as a secondary refinement filter, reflecting a deliberate trade-off between optimality and deployment efficiency.

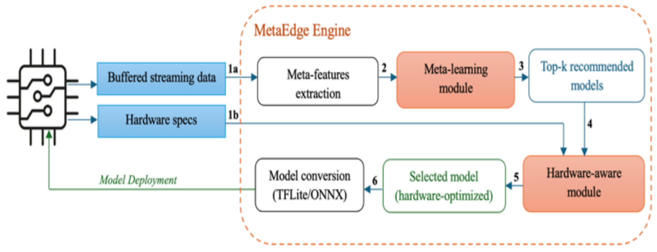


Fig. 4. Workflow of the MetaEdge engine.

Fig. 4 illustrates the detailed workflow of MetaEdge. First, the incoming streaming data is buffered, and the buffered data is sent along with the target hardware specifications to the MetaEdge engine. Then, at the MetaEdge, a two-stage decision process is performed to select the best deployable anomaly-detection model. In the first stage, the meta-learning module extracts the meta-features of the buffered streaming data and recommends the top-k anomaly detectors based only on the data characteristics. In the second stage, the hardware-aware module uses the hardware specifications and converts them into a standardized feature vector to generate probability predictions (confidence scores) for every model in the candidate pool, not just the top-k from the meta-learning module. The confidence score represents the hardware-aware estimate of the probability that a given model is the most suitable option for a given hardware specification. Despite having complete probability information for all 11 models, the system restricts the selection to only the top-k models identified by the meta-learning selection module. Therefore, the hardware-aware selection module selects the model with the highest probability among the filtered top-k. The selected model is then trained on the buffered data and converted to compact models using the required format (TFLite or ONNX). Algorithm 1 presents the integrated two-stage selection process of MetaEdge.

Algorithm 1 Two-Stage Model Selection of MetaEdge.

```
Input: Dataset meta-features M; device specifications D; meta-learner module ML; hardware-aware module HM; K (default K=3)
Output: Selected model $M_{\text{optimal}}$, confidence score conf

// Stage 1: Meta-learning recommendation
Top_K_models ← $ML.recommend\ (M,K=3)$
// Stage 2: Hardware-aware selection
Device_features ← $[D.max_{ram_{mb}}, D.max_{inference_{ms}}, D.max_{size_{mb}}, D.max_{energy_{mj}}, D.requires_{tflite}, D.requires_{onnx}, D.gpu_{available}]$
Device_features_scaled ← $Scaler.transform\ (Device\_features)$
Prob_all ← $HM.predict\_proba\ (Device\_features\_scaled)$
All_predictions ← $Map\ (All\_candidate\_models,\ Prob\_all)$
Filtered_predictions ← $Filter\ (All\_predictions, Top\_K\_models)$
$(M_{optimal}, conf)$ ← $arg\ max\ (filtered\_predictions)$
Return $M_{\text{optimal}}$, conf
```

IV. Experimental Results

This section reports a comprehensive evaluation of the MetaEdge framework along four dimensions: meta-learning-based model selection, hardware-aware model selection, model conversion effectiveness, and deployment feasibility. All experiments were executed on AWS. We used Amazon SageMaker notebook instances to provide a scalable, reproducible setup, and stored models, training data, and results in Amazon S3 for reliable persistence and access. The stream-processing pipeline consumed the skin dataset and implemented a buffer abstraction to emulate cold-start and warm-buffer conditions, enabling a realistic assessment under varied operating scenarios. We tested MetaEdge on four representative edge platforms spanning a wide compute range from a constrained Orange Pi Zero to a high-performance EC2 g4dn.xlarge with a GPU. The two higher-end platforms were instantiated as AWS EC2 instances to reflect practical cloud–edge deployments. The overarching goal is to show that MetaEdge can adapt model selection to device constraints while maintaining acceptable anomaly-detection performance, and to quantify the benefits of its two-stage selection strategy in real-world edge settings.

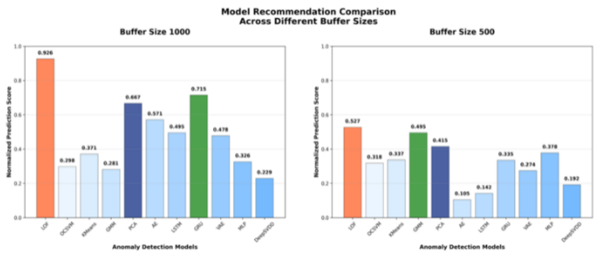


Fig. 5. Model recommendation comparison across different buffer sizes.

We examined how available data volume affects model selection by varying the buffer size (500 vs. 1,000 samples). For each condition, the meta-learning module ranked candidate anomaly detectors and returned the top three. With a 500-sample buffer, it favored LOF, GMM, and PCA; with a 1,000-sample buffer, it favored LOF, GRU, and PCA. As illustrated in Fig. 5, increasing the buffer from 500 to 1,000 samples yielded a 76% average gain in the meta-learner's prediction score. Confidence in LOF rose from 0.527 to 0.926, with corresponding improvements for GRU and PCA. Overall, providing more buffered data makes the meta-learner both more confident and more accurate in its recommendations.

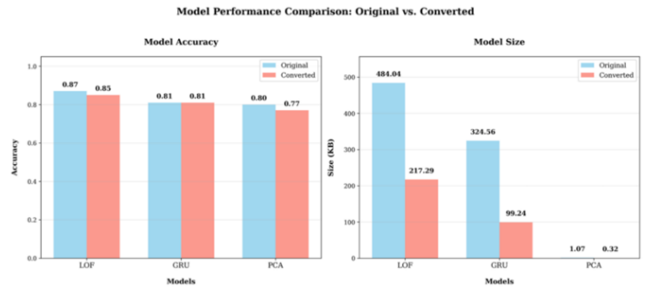


Fig. 6. Model performance comparison (Original vs. Converted).

To evaluate anomaly detection capability, we trained the top three models recommended by the meta-learner: LOF, GRU, and PCA on a 1,000-sample buffer from the skin dataset. We then produced lightweight deployment versions. Where supported, TensorFlow models were exported to .tflite using the official TensorFlow Lite converter; scikit-learn models were converted to ONNX with skl2onnx. In our case, LOF and PCA (scikit-learn) were converted to ONNX, and the GRU model was exported to ONNX via tf2onnx as a fallback mechanism. To validate the effectiveness of these conversions, we compared each converted model with its original counterpart on two metrics: model file size and accuracy (see Fig. 6). Accuracy losses were minimal: LOF decreased from 0.87 to 0.85 (−2.3%), GRU was unchanged at 0.81, and PCA declined from 0.80 to 0.77 (−3.8%). All converted models remained above 75% accuracy. Size compression was substantial: LOF shrank from 484.04 KB to 217.29 KB (−55%), GRU from 324.56 KB to 99.24 KB (−69%), and PCA from 1.07 KB to 0.32 KB (−70%). These reductions, ranging from 55% to 70% make the models well-suited for resource-constrained edge deployments.

TABLE III. THE SPECIFICATIONS OF EDGE DEVICES

Edge Device	Real/ Simulated	Device Specifications
T2.Micro	Simulated	1 vCPU, 1 GB RAM, no GPU; Low-to-moderate network performance [20].
G4dn.Xlarge	Simulated	4 vCPUs, 16 GB RAM, 1× NVIDIA T4 (16 GB VRAM), 125 GB NVMe local storage, up to 25 Gbps networking [21].
Raspberry Pi 5	Real	Quad-core Arm Cortex-A76, 2.4 GHz, VideoCore VII GPU; LPDDR4X-4267 RAM 8 GB; dual 4Kp60 micro-HDMI; Wi-Fi 5 and Bluetooth 5.0 [22].
Orange Pi Zero 2W	Real	quad-core Cortex-A53 (up to 1.5 GHz), Mali-G31 MP2 GPU; 1 GB LPDDR4 ; Wi-Fi 5 and Bluetooth 5.0 [23].

We validated the hardware-aware module by executing it in live deployments across four edge targets. Two were physical devices, a Raspberry Pi 5 (higher-end) and an Orange Pi Zero 2W (resource-constrained), provisioned with AWS IoT Greengrass v2 and packaged via Amazon SageMaker Edge Manager. To widen the hardware spectrum, we also emulated edge profiles on AWS EC2: a t2.micro to represent a low-resource CPU environment and a g4dn.xlarge to represent a GPU-enabled, Jetson-like setting. Device specifications are summarized in Table III. The hardware-aware module leverages each device's specifications to choose the best model from the meta-learner's candidates (LOF, GRU, and PCA), assigning a confidence score to each. Detailed per-device confidence scores for all models are presented in the following subsections.

A. Edge Device 1: Orange Pi Zero

The Orange Pi Zero is one of the resource-constrained platforms, and its confidence scores reflect the framework's ability to adapt to severe hardware limits. LOF is the preferred option at 45% confidence (see Fig. 7), indicating that classical algorithms are best suited to resource-limited devices. PCA is a strong alternative at 25%, underscoring the preference for lightweight, classical approaches. By contrast, the GRU receives 0% confidence, consistent with the view that neural networks are impractical on this class of hardware. Overall, the distribution shows hardware-aware reasoning: traditional ML models dominate the shortlist (LOF 45%, PCA 25%, KMeans 15%), appropriately prioritizing resource-efficient choices for reliable deployment under tight constraints.

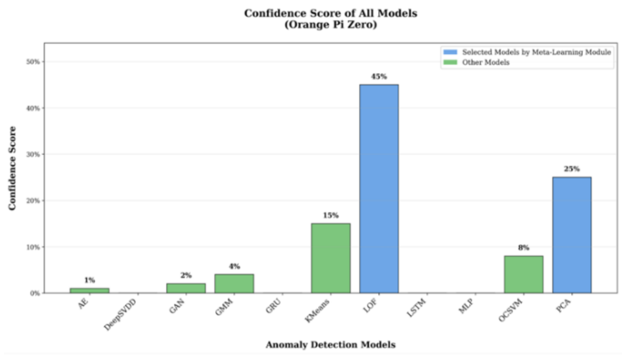


Fig. 7. Confidence scores of the Orange Pi Zero device.

To assess real-time feasibility, we deployed the selected LOF model on the Orange Pi Zero. The inference traces (see Fig. 8) remained stable across 6,000 samples with no anomalous spikes. Latency was tightly centered at 17.36 ms and consistently under 20 ms per sample, meeting real-time requirements for lightweight devices. Chunk-level analysis over 120 windows showed steady performance with only minor fluctuations. While not the fastest platform, the Orange Pi Zero ran LOF inference reliably, confirming that on-device anomaly detection is feasible even on microcontroller-class hardware.

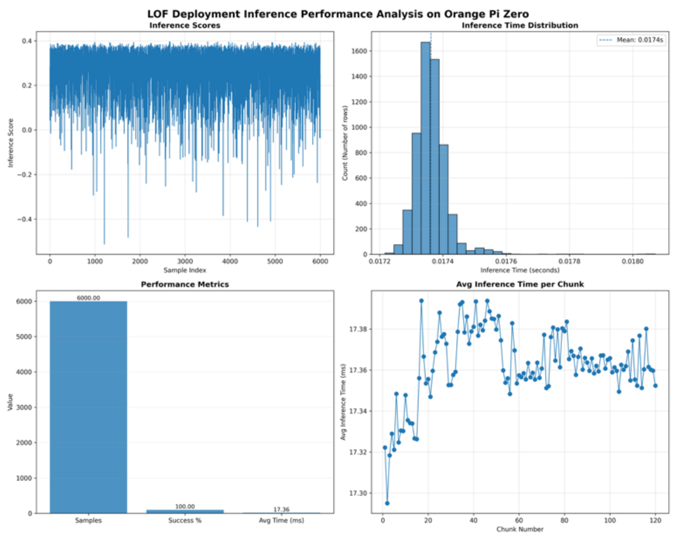


Fig. 8. Inference performance of the Orange Pi Zero device.

Running LOF on the Orange Pi Zero shows that simple, lightweight methods can work well on limited hardware, with 81.0% accuracy, as shown in Fig. 9. Precision is 29.4%, meaning one in three alerts corresponds to a true anomaly, a trade-off that favors responsiveness over selectivity on low-power devices. Recall is 79.1%, so the system catches most of the anomalies useful in settings where missing an event such as equipment faults and security issues is worse than raising extra alerts. Specificity is 81.2%, indicating reasonable discrimination of normal behavior given the algorithm's simplicity and small computational footprint. Overall, the F1 score is 42.8%, and the average precision (AP) is 0.456, reflecting moderate separation between classes under tight resource limits. The confusion matrix (TN = 4,434; FP = 1,026; FN = 113; TP = 427) confirms this profile: more false alarms than some higher-powered platforms, but strong sensitivity that is well suited to applications where missing an anomaly is costlier than investigating extra alerts. Taken together, these results make the Orange Pi Zero and LOF combination a practical fit for distributed sensor networks, remote monitoring, and other IoT deployments where power, cost, and connectivity limitations demand efficient local processing.

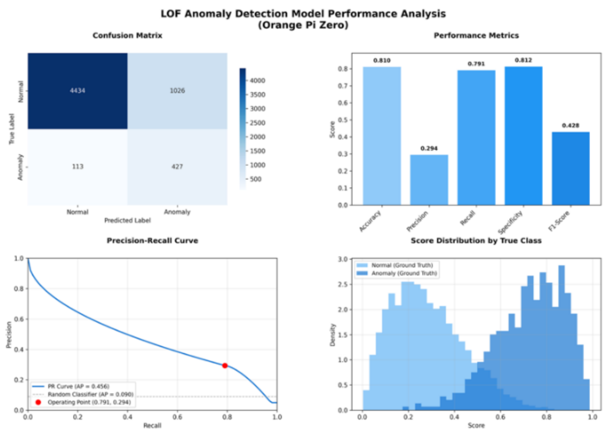


Fig. 9. Performance of a LOF model on Orange Pi Zero: (a) Confusion matrix; (b) Metrics summary; (c) Precision–recall curve; (d) Score distributions by class.

B. Edge Device 2: EC2 T2.Micro

The EC2 t2.micro serves as another resource-constrained target, with limited RAM and tight latency budgets. LOF remains the leading choice at 35% confidence, down from 45% on the Orange Pi Zero, suggesting that the different resources on t2.micro allow closer competition (as shown in Fig. 10). GAN reaches 20% confidence but is excluded because it was not selected by the meta-learning module. Overall, classical options still dominate the viable set PCA at 15% while non-selected candidates such as KMeans (12%) and AutoEncoder (8%) register only moderate confidence.

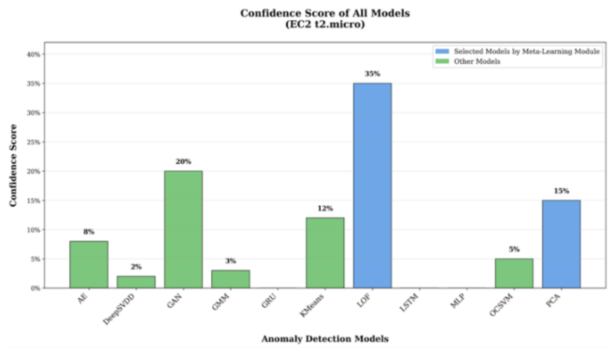


Fig. 10. Confidence scores of T2.micro.

To assess the real-time viability of the chosen LOF model, we deployed it on AWS EC2 t2.micro instances using AWS IoT Greengrass v2. Fig. 11 summarizes the edge-side inference results. The line plot tracks raw anomaly scores across 6,000 streaming samples; scores are normalized to the [0, 1] range and exhibit steady behavior without abrupt spikes. The accompanying histogram reports per-sample latency: the mean inference time was 2.58 ms, with most samples completing under 3 ms. Latencies ranged from 2.34 ms to 12.68 ms indicating efficient and comparatively stable performance on constrained hardware. A chunk-level view (CHUNK_SIZE = 50) further showed uniform throughput across 120 chunks, with no evident degradation or drift. Despite limited CPU and memory, the LOF model sustained stable throughput and accuracy, supporting its suitability for lightweight deployment on t2.micro edge device.

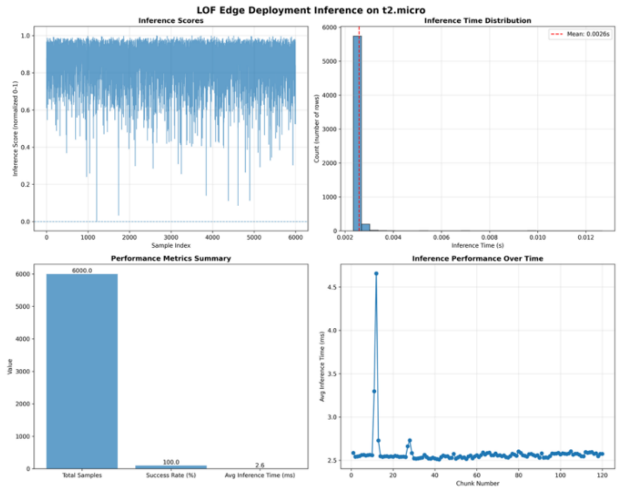


Fig. 11. Inference performance of the T2.micro device.

On an AWS T2.micro instance, the LOF model achieves 76.5% accuracy (see Fig. 12), demonstrating that useful anomaly detection can be achieved on a very low-cost instance. Precision is 23.9%, so about one in four alerts is a real issue, a reflection of the limited CPU and memory on this class of VM, which caps model complexity. Recall comes in at 73.9%, meaning the system catches the most important anomalies, which is valuable when the cost of missing an event could outweigh savings on infrastructure. Specificity is 76.8%, a reasonable level of "normal vs. abnormal" separation for such a constrained setup. Overall, the F1 score is 36.1% and the AP is 0.403, consistent with moderate discriminative power for cost-sensitive use cases. The confusion matrix (TN=4,191; FP=1,269; FN=141; TP=399) shows a higher false-alarm rate, but acceptable recall, which is ideal for cost-sensitive environments that use alerts to prompt lightweight verification.

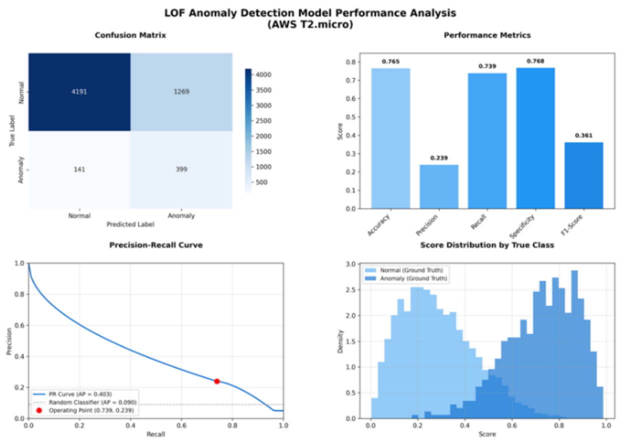


Fig. 12. Performance of a LOF model on T2.micro: (a) Confusion matrix; (b) Metrics summary; (c) Precision–recall curve; (d) Score distributions by class.

C. Edge Device 3: Raspberry Pi 5

The Raspberry Pi 5 marks a clear inflection point where the framework begins to favor deep learning and adapts to support neural models. On this device, a GRU is selected with 28% confidence (see Fig. 13), indicating the system's capacity to accommodate more advanced algorithms as resources increase. This choice supports the framework's principle of scaling model complexity with available compute, enabling neural network–based methods when they become practical and advantageous for anomaly detection. Notably, an LSTM registers 42% confidence among the models that were filtered out. This suggests the hardware-aware selector would have preferred LSTM over GRU for this configuration, but the upstream meta-learning stage, driven by data characteristics, did not choose it. This rigid two-stage pipeline can therefore constrain optimal hardware-aware decisions when the meta-learning and hardware signals diverge.

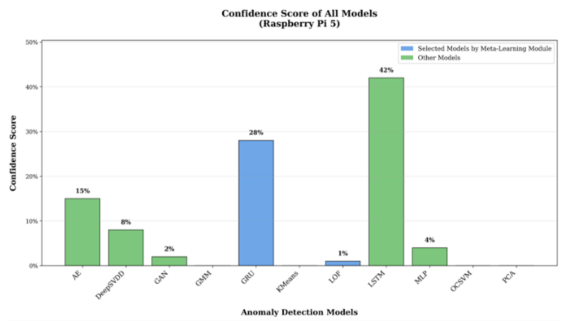


Fig. 13. Confidence scores of the Raspberry Pi 5.

To verify the GRU model's real-time viability, we deployed it on a Raspberry Pi 5. Fig. 14 summarizes the edge-side inference results. Outputs remain stable across all 6,000 samples, with scores confined to the [0, 1] range. Latency is extremely low, an average of 0.076 ms, with the vast majority of inferences finishing under 0.1 ms. A chunk-level analysis over 120 chunks shows flat, highly consistent throughput with no signs of drift or degradation. Overall, the Raspberry Pi 5 executes the GRU model with ultra-low latency and negligible variance, confirming its suitability for real-time anomaly detection at the edge.

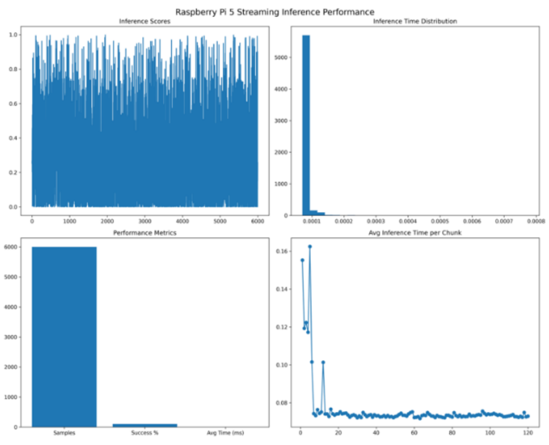


Fig. 14. Inference performance of the Raspberry Pi 5 device.

As shown in Fig. 15, running a GRU model on the Raspberry Pi 5 delivers standout anomaly detection, topping all platforms in our test with 96.6% accuracy. Precision reaches 90.4%, meaning 9 out of 10 alerts are real, which significantly reduces the cost of chasing false alarms. The model's 69.7% recall reflects a deliberate tilt toward accuracy over sensitivity, which is useful in settings where false positives are more costly than the occasional miss. Specificity is an exceptional 99.3%, indicating the system almost never flags normal behavior. Overall performance is strong, with an F1 score of 78.7% and an AP of 0.818. The confusion matrix shows 5,420 correctly identified normal events and 376 correctly identified anomalies, with only 40 false alarms and 164 missed anomalies, evidence of tight control over false positives while capturing a substantial share of true anomalies. These results demonstrate the model's ability to achieve high-performance anomaly detection on edge computing hardware with minimal computational overhead.

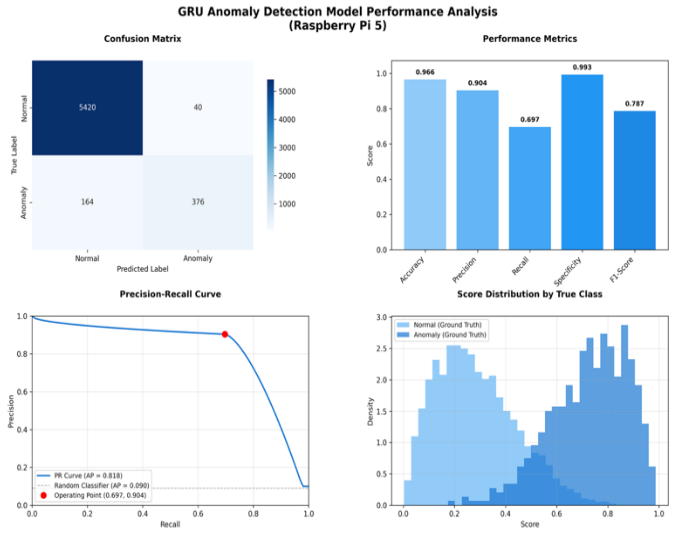


Fig. 15. Performance of a GRU model on Raspberry Pi 5: (a) Confusion matrix; (b) Metrics summary; (c) Precision–recall curve; (d) Score distributions by class.

D. Edge Device 4: EC2 G4dn.Xlarge

The EC2 g4dn.xlarge evaluation shows how the framework performs in a resource-rich, GPU-accelerated setting. As shown in Fig. 16, with ample compute, the system elevates deep learning and selects a GRU with 18% confidence, illustrating its ability to take advantage of advanced hardware. This aligns with the framework's scaling strategy: as constraints ease, more sophisticated neural architectures become preferable. While GRU was prioritized as a top 3 meta-learning choice, LSTM attained 55% confidence, exceeding the GRU by more than threefold, suggesting substantial promise under different dataset conditions. As expected on this class of hardware, traditional ML methods receive negligible confidence (mostly under 5%), avoiding underutilization of the platform. Overall, the high-performance scenario confirms that the framework adjusts model complexity to match available resources, enabling efficient use of advanced edge infrastructure.

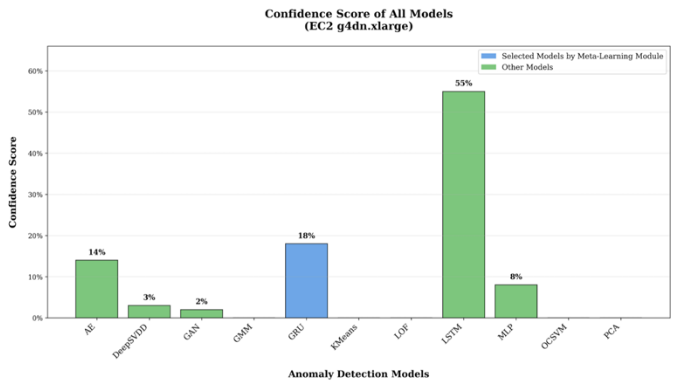


Fig. 16. Confidence scores of G4dn.Xlarge.

To validate the real-time operational feasibility of the selected LOF model, deployments were executed on AWS EC2 instances G4dn.Xlarge using AWS IoT Greengrass V2. The GRU edge deployment inference results for EC2 instance G4dn.Xlarge are shown in Fig. 17. The line chart shows GRU's raw sequence modeling outputs across 6,000 streaming samples, bounded between 0 and 1. The scores remained stable without spikes, reflecting consistent prediction quality. The histogram confirms ultra-stable latency, with most samples completing near 0.07 ms. The narrow range (0.066–0.298 ms) and extremely low variance (std: 0.0053 ms) validate the predictability of GPU execution. Chunk-level latency (120 chunks, 50 samples each) was nearly flat, with no systematic drift, confirming sustained throughput. The GRU model leveraged G4dn.Xlarge's GPU capabilities to deliver extremely consistent, ultra-low-variance inference performance, ideal for precision-critical deployments.

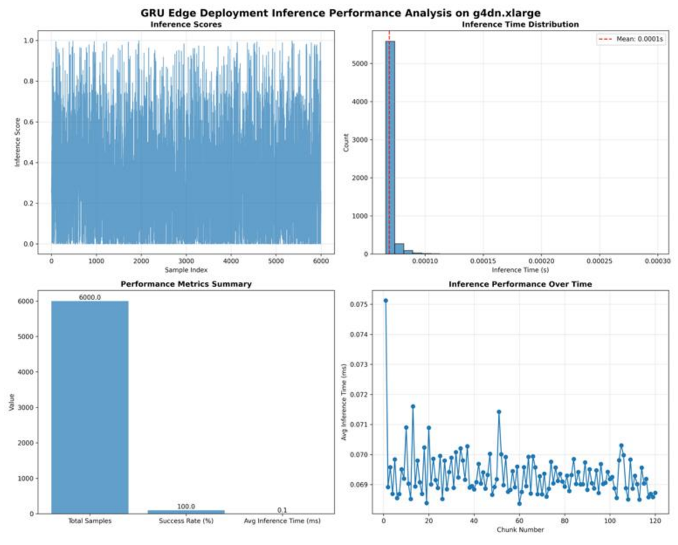


Fig. 17. Inference performance of G4dn.xlarge device.

Running a GRU model on an AWS G4dn.xlarge delivers solid anomaly detection with 89.0% accuracy, highlighting how high-performance hardware helps with complex pattern recognition. Precision is 43.2%, about two of every five alerts are true, a clear step up from our statistical baselines, while keeping throughput high thanks to GPU acceleration. Recall is 69.6%, so the system catches nearly 70% of real anomalies, and specificity is 90.9%, showing it is strong at recognizing normal behavior. Overall, the model posts an F1 of 53.3% and an AP of 0.512, indicating good discrimination across thresholds, as shown in Fig. 18. The confusion matrix shows TN=4,965, FP=495, FN=164, and TP=376, reflecting a balanced trade-off between detection and false alarms. In practice, this GPU-accelerated setup is well-suited to production workloads that need both high performance and straightforward scalability.

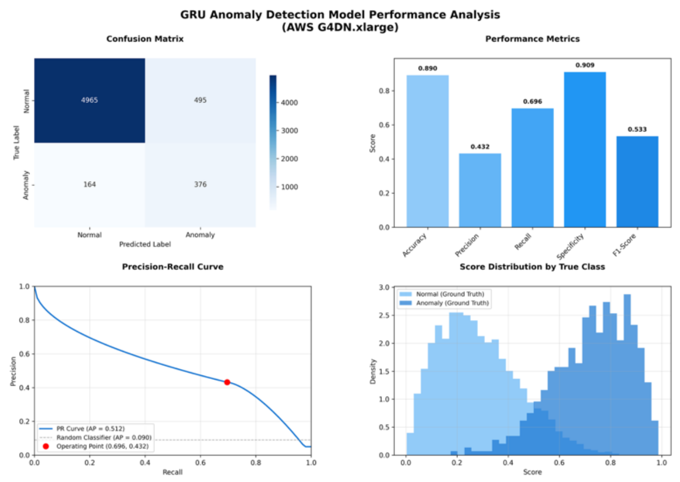


Fig. 18. Performance of a GRU model on G4dn.Xlarge: (a) Confusion matrix; (b) Metrics summary; (c) Precision–recall curve; (d) Score distributions by class.

E. Results Summary

MetaEdge shows strong hardware-aware model selection: its confidence score distributions adapt to the computational limits of the four evaluated devices. The Random Forest meta-selector produces probabilities ranging from 0% for clearly incompatible choices to 55% for the best hardware–model pairs. Concretely, LOF scores 45% confidence on the Orange Pi Zero and 35% on EC2 t2.micro, while GRU reaches 28% on the Raspberry Pi 5 and becomes viable on EC2 g4dn.xlarge, evidence that the framework scales model complexity progressively with available resources. The model conversion process through ONNX standardization achieves remarkable success with only 0–3.8% degradation, while delivering substantial size reductions of 55-70% across all models. These reductions enable practical edge deployment without sacrificing detection capability, effectively bridging the gap between theoretical performance and real-world deployment constraints.

Across deployments, the models deliver strong anomaly detection and validate MetaEdge's two-stage selection strategy. The Raspberry Pi 5 leads all platforms with 96.6% accuracy and 90.4% precision, surpassing all other configurations in this study. The AWS G4dn.xlarge follows with 89.0% accuracy using GPU acceleration, while the LOF model maintains consistent recall on constrained devices (79.1% and 73.9% on the Orange Pi Zero and EC2 t2.micro, respectively). MetaEdge balances performance and deploy ability by first using meta-learning to shortlist high-performing candidates, then applying hardware-aware constraints to ensure each model fits device resources. This produces sensible adaptations: traditional ML is selected for tight edge environments, and deep learning is enabled where hardware permits. The result is superior anomaly detection across diverse edge settings without sacrificing computational efficiency or deployment reliability.

V. Conclusion

This study presents MetaEdge, a hardware-aware framework for model selection that tackles the challenge of deploying effective anomaly detection across heterogeneous edge devices. It combines meta-learning with hardware-aware optimization to enable practical deployment without compromising detection quality. The framework makes high-accuracy model-selection decisions, with confidence scores that provide meaningful assessments of hardware compatibility to guide deployment strategies. MetaEdge uses ONNX standardization for model conversion to enable cross-platform deployment while minimizing accuracy loss, yielding 0–3.8% degradation and a 55–70% reduction in model size.

Experimental results demonstrate superior anomaly-detection performance across diverse hardware configurations. The hardware-aware stage consistently identifies optimal model–hardware pairings, and the deployed models exhibit strong detection performance. On a Raspberry Pi 5, MetaEdge achieves 96.6% accuracy and 90.4% precision under strict computational constraints, outperforming the other tested implementations. On an AWS g4dn.xlarge instance, GPU acceleration yields 89.0% accuracy. The LOF model maintains stable recall on resource-constrained devices (79.1% and 73.9%), providing a reliable baseline for such deployments.

MetaEdge demonstrates that automated, hardware-aware model selection can jointly identify models from data meta-features and align them with device constraints. The framework operates across heterogeneous platforms from ultra-constrained edge nodes to GPU-accelerated systems, enhancing on-device anomaly detection while adhering to strict resource budgets. Beyond empirical gains, MetaEdge offers methodological guidance for distributed anomaly detection and clarifies the trade-offs inherent in hardware-aware optimization. It establishes a robust foundation for future adaptive edge computing systems, capable of intelligently adjusting their performance based on available computational resources and deployment constraints.

In practice, MetaEdge supports real-world deployment by providing a systematic workflow for selecting accurate models that satisfy explicit resource budgets across diverse devices, reducing trial-and-error when transitioning from experimental evaluation to on-device operation. As with any deployment-oriented framework, MetaEdge operates within the assumptions of the profiled model space and observed device conditions; however, its modular design allows these profiles to be incrementally expanded and updated as deployment scale, anomaly characteristics, and hardware diversity evolve. To further strengthen applicability at scale, future work will extend the profiling process and model registry to broader model families and a wider range of devices, support lightweight online learning to maintain performance under concept drift and changing anomaly patterns, enable model personalization to device- and context-specific conditions, and explore cross-device collaborative learning to improve robustness across heterogeneous edge deployments, while reducing runtime overhead through offline profiling and caching.

Acknowledgment

This work is derived from a research grant funded by the Research, Development, and Innovation Authority (RDIA), Kingdom of Saudi Arabia, with grant number 12615-iu-2023-IU-R-2-1-EI-. The authors gratefully acknowledge the anonymous reviewers for their insightful feedback and valuable suggestions.

References

[1] M. G. Sarwar Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, “Machine Learning at the Network Edge: A Survey,” ACM Computing Surveys (CSUR), vol. 54, no. 8, Oct. 2021, doi: 10.1145/3469029.

[2] M. M. H. Shuvo, S. K. Islam, J. Cheng, and B. I. Morshed, “Efficient Acceleration of Deep Learning Inference on Resource-Constrained Edge Devices: A Review,” Proceedings of the IEEE, vol. 111, no. 1, pp. 42–91, Jan. 2023, doi: 10.1109/JPROC.2022.3226481.

[3] P. Schneider and F. Xhafa, “Anomaly detection: Concepts and methods,” Anomaly Detection and Complex Event Processing over IoT Data Streams, pp. 49–66, Jan. 2022, doi: 10.1016/B978-0-12-823818-9.00013-4.

[4] S. Jadhav and A. Kulkarni, “Comprehensive Survey on Detection of Anomalies in Edge Computing Network and Deep Learning Solutions,” 2025, doi: 10.5220/0013344100004646.

[5] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 67–82, 1997, doi: 10.1109/4235.585893.

[6] M. Garouani, A. Ahmad, M. Bouneffa, M. Hamlich, G. Bourguin, and A. Lewandowski, “Using meta-learning for automated algorithms selection and configuration: an experimental framework for industrial big data,” J Big Data, vol. 9, no. 1, pp. 1–20, Dec. 2022, doi: 10.1186/S40537-022-00612-4/TABLES/8.

[7] S. M. Raza, S. M. H. Abidi, M. Masuduzzaman, and S. Y. Shin, “Lightweight deep learning for visual perception: A survey of models, compression strategies, and edge deployment challenges,” Neurocomputing, vol. 656, p. 131357, Dec. 2025, doi: 10.1016/J.NEUCOM.2025.131357.

[8] A. Chatterjee and B. S. Ahmed, “IoT anomaly detection methods and applications: A survey,” Internet of Things, vol. 19, p. 100568, Aug. 2022, doi: 10.1016/J.IOT.2022.100568.

[9] D. R. Patrikar and M. R. Parate, “Anomaly detection using edge computing in video surveillance system: review,” Int J Multimed Inf Retr, vol. 11, no. 2, pp. 85–110, Jun. 2022, doi: 10.1007/S13735-022-00227-8.

[10] Y. Ying, J. Duan, C. Wang, Y. Wang, C. Huang, and B. Xu, “Automated Model Selection for Time-Series Anomaly Detection,” Aug. 2020, doi: 10.1145/nnnnnnn.nnnnnnn.

[11] Z. Zhao, K. Wang, N. Ling, and G. Xing, “EdgeML: An AutoML Framework for Real-Time Deep Learning on the Edge,” IoTDI 2021 - Proceedings of the 2021 International Conference on Internet-of-Things Design and Implementation, pp. 133–144, May 2021, doi: 10.1145/3450268.3453520.

[12] E. Tang, B. Yang, and X. Song, “MetaOOD: Automatic Selection of OOD Detection Models,” Transactions on Machine Learning Research, vol. 2025, 2025, Accessed: Sep. 28, 2025. [Online]. Available: https://github.com/yqin43/metaood.

[13] Z. Li, H. Li, and L. Meng, “Model Compression for Deep Neural Networks: A Survey,” Computers 2023, Vol. 12, Page 60, vol. 12, no. 3, p. 60, Mar. 2023, doi: 10.3390/COMPUTERS12030060.

[14] Y. Xue, “Comparison Of Conventional and Lightweight Convolutional Neural Networks for Image Classification,” Highlights in Science, Engineering and Technology, vol. 38, pp. 988–993, Mar. 2023, doi: 10.54097/HSET.V38I.5986.

[15] H. Lee, S. Lee, S. Chong, and S. J. Hwang, “HELP: Hardware-Adaptive Efficient Latency Prediction for NAS via Meta-Learning,” 2021, Neural information processing systems foundation. Accessed: Sep. 28, 2025. [Online]. Available: https://pure.kaist.ac.kr/en/publications/help-hardware-adaptive-efficient-latency-prediction-for-nas-via-m.

[16] R. Das and T. Luo, “LightESD: Fully-Automated and Lightweight Anomaly Detection Framework for Edge Computing,” 2023 IEEE International Conference on Edge Computing and Communications (EDGE), pp. 150–158, Jul. 2023, doi: 10.1109/EDGE60047.2023.00032.

[17] M. J. C. S. Reis and C. Serôdio, “Edge AI for Real-Time Anomaly Detection in Smart Homes,” Future Internet 2025, Vol. 17, Page 179, vol. 17, no. 4, p. 179, Apr. 2025, doi: 10.3390/FI17040179.

[18] N. Rashid, R. Mehmood, F. Alqurashi, S. Alqahtany, and J. M. Corchado, “ASAD: A Meta Learning-Based Auto-Selective Approach and Tool for Anomaly Detection,” IEEE Access, 2024, doi: 10.1109/ACCESS.2024.3524908.

[19] A. Liaw and M. Wiener, “Classification and Regression by randomForest,” vol. 2, no. 3, 2002, Accessed: Oct. 04, 2025. [Online]. Available: http://www.stat.berkeley.edu/.

[20] “Amazon EC2 T2 Instances – Amazon Web Services (AWS).” Accessed: Oct. 06, 2025. [Online]. Available: https://aws.amazon.com/ec2/instance-types/t2/.

[21] “Now Available – EC2 Instances (G4) with NVIDIA T4 Tensor Core GPUs | AWS News Blog.” Accessed: Oct. 06, 2025. [Online]. Available: https://aws.amazon.com/blogs/aws/now-available-ec2-instances-g4-with-nvidia-t4-tensor-core-gpus/?utm_source=chatgpt.com.

[22] “Computers - Product Information Portal - Raspberry Pi.” Accessed: Oct. 06, 2025. [Online]. Available: https://pip.raspberrypi.com/categories/505-computers.

[23] “Orange Pi - Orangepi.” Accessed: Oct. 06, 2025. [Online]. Available: http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-Zero-2W.html.