A Self-Calibrating Agentic AI Framework for Autonomous Edge Resource Allocation

Fin Gentzen, Marla Grunewald, Iulisloi Zacarias, Mounir Bensalem and Admela Jukan
Institut für Datentechnik und Kommunikationsnetze
Technische Universität Braunschweig, Germany
Email: {f.gentzen, marla.grunewald, i.zacarias, mounir.bensalem, a.jukan}@tu-bs.de

***Abstract*—Large Language Models (LLMs) are increasingly deployed as autonomous agents, transitioning from static conversational interfaces to dynamic systems capable of complex reasoning, tool execution, and decision-making. However, the operational reliability of these agentic AI systems is fundamentally challenged by the absence of reliable ground truth in open-ended environments and the risk of increasing operational drift over time. To address this challenge, we propose and experimentally evaluate an agentic AI framework, designed to enforce autonomous integrity within LLM-driven systems. We design a self-calibration mechanism that mitigates drift and dynamically approximates ground truth by incorporating an ARIMA forecaster, without requiring continuous human oversight. To demonstrate the effectiveness and reliability of our methodology, we apply it to the complex domain of profiling the resource usage of zero-knowledge workloads in edge computing networks. Experimental results show that the proposed self-calibrating agentic framework successfully profiles the zero-knowledge workloads, achieving a higher accuracy than baseline LLM agents by 91.7% for resource usage prediction and improving the prediction speed by 71.7% compared to pure profiling, establishing a robust foundation for deploying autonomous AI in decentralized infrastructures. Furthermore, the ground truth generation using the proposed ARIMA leaping algorithm is 52% faster than a standard ARIMA forecasting algorithm, while achieving the same accuracy.**

***Index Terms*—Agentic AI, Self-Calibration, Edge Network, AI Workload**

I. Introduction

THE exponential growth of distributed Artificial Intelligence (AI) applications is forcing network service providers to rethink their network architectures and deployment plans. AI training and inference processes, once centralized in large data centers, are increasingly migrating to the network edge. Placing AI workloads at the network's edge offers compelling advantages for time-sensitive and privacy-concerned applications. Especially lightweight AI workloads are expected to be placed closer to the end-user, thereby distributing the computing tasks along the computing continuum, whereas resource-hungry workloads are still likely to be placed in data centers. For an efficient implementation of the compute continuum, edge devices are inherently resource-constrained devices in a highly heterogeneous landscape [1] and, as such, present a challenge for efficient workload placement. To address this challenge, the rapid evolution of autonomous agentic AI is giving rise to agentic edge intelligence [2].

Despite significant benefits, agentic edge intelligence exposes a fundamental operational problem. In contrast to homogeneous cloud data centers, where elastic resource pools (e.g., CPU, GPU, memory, and storage) are available, (far-) edge environments are characterized by hardware heterogeneity, limited computational capacity, and constrained memory and storage. Correctly allocating resources for AI workloads across the computing continuum requires precise knowledge of resource consumption in advance, which is typically unavailable. In edge environments, resource consumption can drift during execution, for example, by requiring fewer or more computing resources compared to the initial estimation; in one case it can lead to resource underutilization, while in the other, the AI workload might fail, by abruptly finishing its execution without producing any results. The workload would then likely need to be reallocated to computing nodes capable of handling it, and processing would restart. This situation may lead to Service Level Agreements (SLAs) breaches due to increased processing latency and low user satisfaction, and potentially disastrous outcomes in time-sensitive systems. What is needed is a precise estimation of the resource consumption of AI workloads, backed by the reliable ground-truth data to validate estimation algorithms and as well as self-calibration mechanisms to counteract runtime drift.

This paper proposes a novel self-calibration agentic AI framework that can efficiently address the problem of edge resource allocation for AI workloads with zero-knowledge *a priori*. It assumes that AI workloads arrive at edge devices as black-box executables paired with a dataset which may vary in size and format. Upon arrival, no accompanying metadata describing their memory footprint, CPU and GPU requirements, or execution time is available. Our solution is autonomous, as it does not require human oversight while it combines Large Language Model (LLM)-based prediction with empirical telemetry observation, dynamically approximating ground truth. An AI Workload Agent orchestrates four modules: i) LLM Zero-Shot Estimation that evaluates and predicts the workload resource usage by performing static code analysis; ii) Active Profiling, that combines partial empirical evaluation of the workloads with an ARIMA-based forecasting engine to extrapolate the collected measurements to predict full-scale resource consumption; iii) LLM + Retrieval Augmented Generation (RAG) Estimation, that uses previous knowledge generated by the system to predict system load with higher accuracy without empirical evaluation of the workload; and iv) Re-Profile and Calibrate module that is triggered when the system fails to fulfill any of the environment constraints.

The novel contributions of this study can be summarized as follows:

- A self-calibrating agentic AI architecture for zero-knowledge workload profiling, integrating a reasoning controller, a policy and constraints manager, and four complementary profiling modules that can be invoked independently or concurrently. To the best of our knowledge, this is the first framework that autonomously generates and refines its own ground truth for edge resource allocation.
- A formal system model for bootstrap profiling, covering both discrete-memory and unified-memory architectures, which employs $M/G/\infty$ queuing for CPU modeling, occupancy-based GPU modeling, and ARIMA-based forecasting to extrapolate sandbox executions to full-scale resource footprints.
- An adaptive parameter search algorithm with three leaping strategies that navigates the sample/epoch configuration space under strict time budgets, terminating early once forecast confidence exceeds validated thresholds.
- An experimental framework with open benchmark dataset of 53 profiled AI workloads spanning eight model architectures, six datasets, and three heterogeneous hardware platforms (Raspberry Pi 5, NVIDIA Jetson Thor, GPU workstation), released to support full reproducibility of research and for usage of data in the research community.
- A comprehensive experimental evaluation shows how our proposed agentic framework reduces prediction error from over 200% MAPE to single-digit MAPE for well-covered workload classes, while it is still faster than classical ARIMA-only workload estimators.

The rest of the paper is organized as follows. Section II presents related work. Section III describes the proposed architecture, illustrating the hierarchical agentic AI workflow, the developed resource prediction workflows, and the tools employed. Section IV presents a theoretical model of the *Bootstrap Profiling* method together with the problem formulation. In Section V, we present the dataset that was created in this study. Section VI presents and discusses the analytical and experimental results. Section VII concludes the paper and provides directions for further research.

II. Related Work

A. LLMs and Agentic AI in Network Management

Industry forums have extensively demonstrated how Generative artificial intelligence (GenAI) reshaped network monitoring and orchestration [3], [4], [5]. Moving beyond basic conversational interfaces, these models translate high-level intents into dynamic network configurations [6], [7], [8]. Furthermore, autonomous entities have been embedded to mitigate network outages [9], and modern frameworks employ RAG to bind generated parameters to external verifiable knowledge bases, significantly reducing model hallucinations [10], [11].

In line with these trends, our framework relies on the complex reasoning capabilities of LLMs and RAG to orchestrate tasks and translate high-level constraints autonomously. However, our operational assumptions differ significantly. Most existing applications assume centralized environments with virtually unbounded computing power or rely on infrastructure knowledge that is confined to explicitly declared models, such as YANG or TOSCA. What these papers are not addressing is the challenge of orchestrating zero-knowledge AI executables arriving at highly constrained and heterogeneous edge architectures without any accompanying metadata. We directly address this gap by proposing a self-calibrating agentic AI architecture that integrates a reasoning controller to autonomously generate and refine its own ground truth without human oversight.

B. Theoretical Modeling of AI Telemetry Data

To correctly allocate resources across the computing continuum, establishing an analytical foundation for resource consumption is critical. Earlier works have proposed theoretical abstractions to predict hardware loads. For instance, statistical queue models have been successfully utilized for reliable forecasting of future CPU consumption based on request arrival patterns [12]. Similarly, in the domain of hardware acceleration, studies have formalized the analysis of resource utilization and occupancy parameters on GPUs [13].

Based on these studies, however applied in different contexts, we apply mathematical queueing and occupancy theories to abstract and estimate physical hardware utilization during task execution. Our methodology extends the existing literature majorly, by integrating both assumptions together with constraints for memory, and using this combination to describe a mathematical telemetry benchmark system for AI workloads.

C. Telemetry-Driven Workload Forecasting and Profiling Models, and Reproducibility

Deploying workloads accurately across the edge continuum requires anticipating resource consumption, a challenge often tackled through statistical forecasting and deep learning. Traditional machine learning models have been successfully used to predict virtual machine execution times and to model network interference impacts [14], [15]. For extrapolating short-term telemetry observations, time series forecasting methods such as ARIMA are commonly deployed [16], [17]. More recently, deep learning ensembles and lightweight feedforward neural networks have been developed to capture temporal and spatial patterns for predicting execution times across edge-fog-cloud architectures [18], [19]. Active telemetry profiling is also widely utilized to evaluate multi-criteria cost functions dynamically and swap between complex and lightweight models to prevent hardware overload [20].

We operate similarly by relying on active profiling and ARIMA-based forecasting engines to extrapolate partial empirical observations into full-scale predictions. The key difference is that existing methods are relatively slow and need to be fed a lot of data to make sophisticated forecasts. They cannot speed up the forecasting process based on dynamic accuracy changes by filtering the forecasting data. We overcome this limitation by developing a leaping strategy that significantly speeds up the ARIMA forecasting time, while achieving a very comparable accuracy.

A review of the literature reveals a distinct lack of comprehensive open datasets for evaluating autonomous edge resource allocation. Most related frameworks validate their estimation algorithms using proprietary network traces, constrained simulations, or legacy datasets that do not reflect the diverse operational drift of modern containerized AI applications on physical edge hardware.

While we perform rigorous experimental evaluations similarly to the existing literature, our approach differs by prioritizing full transparency and physical hardware heterogeneity across the compute continuum. The existing literature does not address the community need for reliable ground-truth telemetry spanning multiple modern model architectures and diverse physical edge devices. We explicitly address this missing literature in our fourth contribution by releasing an experimental framework and an open benchmark dataset of 53 profiled AI workloads spanning eight model architectures, six datasets, and three heterogeneous hardware platforms, thereby supporting the full reproducibility of research in agentic edge intelligence.

D. Edge Resource Orchestration and Multi-Agent Systems

To manage the limited capacity and significant hardware heterogeneity of the edge continuum, multi-agent reinforcement learning (MARL) is frequently used for task offloading and resource scheduling [21], [22], [23]. In these systems, distributed agents optimize policies for dynamic load balancing [24], and digital twins are increasingly used to assist in decentralized environments [25]. To overcome the inflexibility of pure reinforcement learning, newer hybrid architectures combine classical optimization techniques with LLM-based reasoning for adaptive agent placement [26] and serverless distributed inference scaling [27].

While our framework shares the ultimate goal of minimizing SLAs breaches in dynamic edge environments, we differ fundamentally in our problem formulation and operational constraints. Existing MARL approaches function primarily as decision-making policy engines for the resource allocation process. These systems assume that the environmental state, such as the resource footprint of a workload, can be observed or learned over many episodes. Consequently, they suffer from a severe cold-start problem and require extensive offline pre-training, rendering them ineffective when confronted with entirely unseen, black-box executables. Furthermore, when workloads experience runtime drift, MARL agents treat this as environmental non-stationarity, necessitating slow and computationally expensive online re-convergence that violates strict edge time budgets. Because these works focus on the placement policy rather than the precursor problem of zero-knowledge resource profiling, they serve as a different class of system. We directly address this estimation bottleneck by proving that our agentic architecture shortening the cold-start phase severely, generating highly accurate resource forecasts under strict time budgets while having no initial information about the workloads at all. Our solution can be used by other MARL approaches to enrich the state information and make a sophisticated resource allocation decision.

III. Architecture

Figure 1 shows the proposed architecture. It includes an agent, the AI Workload Agent, consisting of an A2A Interface, a Reasoning LLM and four different main modules, that can be called independently or concurrently by the LLM. The final output of this AI Workload Agent is the predicted resource usage of an (AI) workload in a distributed edge cluster. In our implementation, an (AI) workload is provided to the system in JSON format, pointing to the Python file in which the AI task is described and a pointer to the dataset that is used for the training process. Together with the JSON file, the reasoning controller is connected to the Policy and Constrains Manager, over which the resource prediction process can be configured.

The key four main modules of the AI Workload Agent provide the reasoning controller with distinct methodological approaches to profile the incoming workloads, i.e.,

1) LLM Zero-Shot Estimation: It utilizes the reasoning LLM to perform a direct, zero-shot profiling of the AI workload. By analyzing the provided Python script and the characteristics of the linked dataset, the model infers a preliminary forecast of the resource usage. Hereby, it relies solely on its pre-trained knowledge base without any prior execution.

2) Active Profiling (Ground Truth): To establish a reliable ground truth, this module employs an ARIMA forecasting approach based on actual execution metrics. The AI workload, consisting of the script and the data, is packaged into an isolated container and executed within a secured sandbox environment. To ensure a time-efficient execution, it is limited to a subset of the real-world machine learning parameters, such as a reduced number of epochs or a smaller sample size. During this execution, the workload is continuously benchmarked. The gathered historical benchmark data is then fed into the ARIMA model to forecast the resource consumption of the next execution batch. If this prediction converges and falls inside a specific predefined bound, the benchmarking process is halted and the final prediction is calculated. If not, the newly acquired benchmark data is subsequently fed back into the ARIMA model to serve as more data for refining the prediction of the next batch. Finally, the converged ARIMA forecast, together with its corresponding input parameters, is stored persistently in a dedicated profiling knowledge base.

3) LLM + RAG Estimation: It enhances the initial estimation by employing a RAG mechanism. The stored forecasts from the ARIMA model serve as the foundational RAG database. Using the input JSON, which points to the Python file and the dataset, the LLM queries this database for similar past executions. By incorporating this historical ground-truth data into the prompt context, the system effectively overcomes the issues of LLM hallucinations.

4) Re-Profile and Calibrate: The outputs generated by the three previously described workflows are all forwarded to the Constraints Verification agent. This agent checks if the formulated predictions comply with the predefined rules in the Policy and Constraints Manager. If a violation is detected (i.e., the policies and constraints are not ok), this information is routed to the Re-Profile and Calibrate function. This mechanism is capable of re-trigger the orchestrator to reevaluate

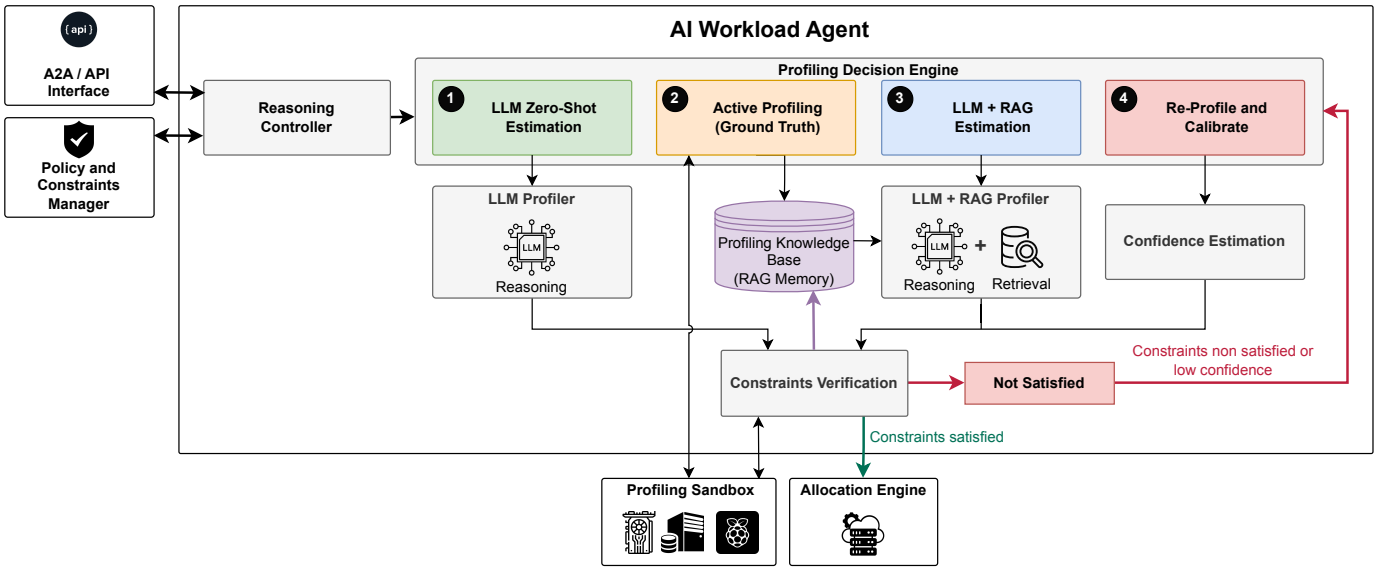


Figure 1: System workflow architecture

with additional information. Based on the past executions, the objective hereby is to gather more extensive input data or to establish a new, more accurate ground truth for calibration.

To illustrate the orchestration of these functionalities, an example workflow can be described as the following. Initially, the policies are configured by the user; for instance, by defining a maximum profiling duration of one minute and setting a nearest-neighbor distance threshold. An incoming AI workload is then sent to the A2A Interface. Subsequently, the reasoning controller analyzes the incoming message and determines the optimal routing to one or more of the available tools. Assuming the profiling knowledge base is initially empty, the controller is aware that a RAG-based approach will not yield results. Therefore, it forwards the task concurrently to the first tool (LLM Zero-Shot Estimation) and the second tool (Active Profiling). The first tool provides an answer very rapidly, albeit with lower accuracy. Meanwhile, the second tool begins the sandboxed execution but exceeds the configured one-minute time limit. Due to this policy constraint, the answer from the first tool is selected for the immediate prediction, because the second tool is taking too long. However, the second tool continues its operation asynchronously until it finishes the execution, subsequently feeding the gathered ARIMA forecasting data into the profiling knowledge base. When the next AI workload is posted to the A2A Interface, the reasoning controller performs a k-Nearest Neighbors search within the knowledge base. If the defined distance threshold is met, indicating that a similar workload was already profiled, the task is exclusively routed to the third tool (LLM + RAG Estimation). If the threshold is not met, the controller falls back to utilizing the first and second tools concurrently. As established before, if the second tool fails to finish within the required timeframe, the fast answer from the first tool is used; otherwise, the accurate result from the second tool is preferred. This iterative process continues, thereby continuously enriching the knowledge base and improving the overall prediction accuracy of the framework over time.

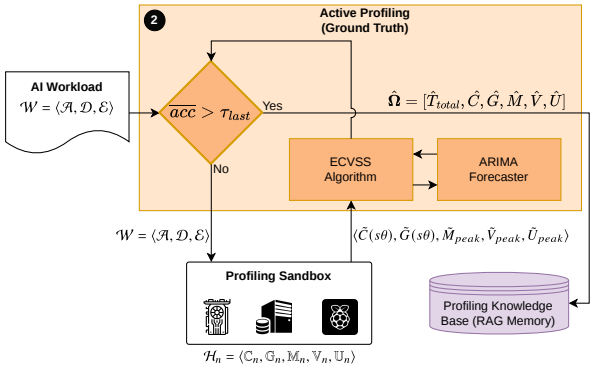


Figure 2: Data flow of the active profiling module and ground truth generation making use of empirical evaluation of AI workloads in a sandboxed environment.

IV. System Model and Ground Truth Formulation

This section outlines the formal mathematical formulation for the resource consumption of containerized AI workloads on edge nodes prior to full deployment. Let an incoming AI workload be defined as a tuple $\mathcal{W}$:

$$\mathcal{W} = \langle \mathcal{A}, \mathcal{D}, \mathcal{E} \rangle, \tag{1}$$

where variable $\mathcal{A}$ represents the algorithm or Python execution graph, while $\mathcal{D} = \{d_k | \forall k \in [1, N]\}$ denotes the complete target dataset (e. g. csv, images, text sequences), where d_k is the k^{th} data point. Finally, $\mathcal{E} = \{\mathcal{E}_\xi | \forall \xi\}$ denotes a set of execution environment constraints, i.e. $\mathcal{E}_1 = 1$ if the node supports CUDA, and 0 otherwise. The *Reasoning Controller* (see Figure 1 routes the AI workload $\mathcal{W}$ to the *Active Profiling* module depicted in Figure 2 to generate ground truth data.

The bootstrap process creates a sampled execution environment. Let $s \in (0, 1]$ denote the bootstrap sampling ratio. The samples are randomly chosen, where we define a random

Table I: Summary of Key Mathematical Notations

Symbol	Description
Hardware & Resource Constraints	
$\mathcal{H}_n$	Set of resource constraints for edge node n
$\mathbb{C}_n, \mathbb{G}_n$	CPU capacity (cores/frequencies) and GPU throughput (TFLOPS)
$\mathbb{M}_n, \mathbb{V}_n, \mathbb{U}_n$	System RAM, VRAM, and Total Unified Memory capacity (GB)
$c(t), g(t)$	Instantaneous CPU and GPU utilization at time t
$m(t), v(t), u(t)$	Instantaneous RAM, VRAM, and unified memory utilization at time t
$\hat{M}_{peak}, \hat{V}_{peak}, \hat{U}_{peak}$	Predicted full-scale peak system RAM, VRAM, and Unified Memory
$\delta_k(s)$	Monotonically decreasing memory safety margin for memory pool $k \in \{m, v, u\}$
Workload & Profiling Metrics	
$\mathcal{W}$	Incoming AI workload tuple $\langle \mathcal{A}, \mathcal{D}, \mathcal{E} \rangle$
$\mathcal{E}$	Set of execution environment constraints
N, s	Total dataset size and bootstrap sampling ratio $s \in (0, 1]$
$\mathcal{D}_s, B$	Sub-sampled dataset of size $\lceil s \cdot N \rceil$ and inference batch size
$T_{prof}(s)$	Total profiling overhead time (T_{init} + T_{load} + T_{exec})
$T_{init}, T_{load}, T_{exec}$	Static container initialization, model loading, and dynamic execution time
$\tau(x), \bar{\tau}_B$	Execution latency for sample x and mean batch processing time
$\mathbf{R}(t)$	Multi-variate time-series matrix of resource utilization
θ, γ	Telemetry sampling period and number of periods in the bootstrap phase
Φ	Extrapolation function mapping telemetry to full-scale footprint
$\hat{\mathbf{\Omega}}$	Full-scale predicted workload footprint $[\hat{T}_{total}, \hat{C}, \hat{G}, \hat{M}, \hat{V}, \hat{U}]$
CPU & GPU Modeling	
$C(t), G(t)$	Instantaneous CPU utilization and GPU occupancy ratio
E	Total number of incoming events in the CPU queue
$\tilde{C}(s\theta), \tilde{G}(s\theta)$	Average CPU and GPU utilization over interval $[s\theta, (s+1)\theta]$
a_e, d_e, p_e	Arrival, departure, and processing time of $M/G/\infty$ queue event e
T_{SM}, R_{SM}	Maximum resident threads and registers per Streaming Multiprocessor
$T_b(t), B_b(t)$	Number of active resident threads and active blocks per kernel
$Z_b(t), R_b(t)$	Active threads per block and registers required per block
L	Total number of observed GPU kernel executions
g_ℓ, u_ℓ, v_ℓ	Occupancy level, start time, and completion time of GPU kernel ℓ
Forecasting & Algorithm Constraints	
X_t, Y_t	Discrete-time historical sequence and its stationary differenced series
p, d, q	ARIMA autoregressive, differencing, and moving average orders
$\mathbf{\Phi}, \mathbf{\Theta}$	Estimated ARIMA parameter vectors for autoregressive and moving average
$\mathcal{M}$	Configuration matrix of dimensions $R \times C$ (epochs × sample size)
T_{max}	Global computational time budget limit for parameter search
τ_{next}, τ_{last}	Local and global accuracy thresholds for surrogate evaluation
ϱ	Target accuracy for parameter search
I	Multi-strategy index dictating the traversal leap behavior

mapping $\varphi : [1, s \cdot N] \rightarrow [1, N] | i \rightarrow \varphi(i)$. We construct a sub-sampled dataset $\mathcal{D}_s = \{x_{\varphi(1)}, ..., x_{\varphi(s \cdot N)}\} \subset \mathcal{D}$ such that $|\mathcal{D}_s| = \lceil s \cdot N \rceil$. The total profiling overhead time, $T_{prof}(s)$, is composed of a static container initialization overhead (T_{init}), model loading time (T_{load}), and the dynamic execution time over the sampled dataset (T_{exec}):

$$\begin{aligned} T_{prof}(s) &= T_{init} + T_{load} + T_{exec}, \quad \text{where} \\ T_{exec} &= \sum_{i=1}^{\lceil s \cdot N \rceil} \tau(x_{\varphi(i)}) \end{aligned} \tag{2}$$

where $\tau(x_{\varphi(i)})$ is the execution latency for the i^{th} data sample $x_{\varphi(i)} \in \mathcal{D}_s$. Assuming Independent and Identically Distributed (I.I.D.) samples and a batch size B, the dynamic term can be approximated by $\lceil \frac{s \cdot N}{B} \rceil \cdot \bar{\tau}_B$, where $\bar{\tau}_B$ is the mean batch processing time, which will give us an updated execution time, given as.

$$T_{exec} = \sum_{i=1}^{\lceil \frac{s \cdot N}{B} \rceil} \tau(x_{\varphi((i-1)B+1)}, ..., x_{\varphi((i-1)B+B)}) \tag{3}$$

To accurately map this dynamic execution time to physical node saturation, we must formalize the underlying hardware environment. We differentiate between systems with unified memory and systems with discrete memory (system RAM plus VRAM), as edge devices can range from simple Raspberry Pi boards over NVIDIA Jetson systems to standard PCs with a graphics card. We denote by $\mathcal{H}_n$ the set of resource constraints of node n, see Figure 2.

$$\mathcal{H}_n = \langle \mathbb{C}_n, \mathbb{G}_n, \mathbb{M}_n, \mathbb{V}_n, \mathbb{U}_n \rangle \tag{4}$$

where $\mathbb{C}_n$ is CPU capacity (cores/frequencies) of node n, $\mathbb{G}_n$ is GPU throughput (TFLOPS), $\mathbb{M}_n$ is system RAM in GB, and $\mathbb{V}_n$ is VRAM capacity in GB as well. For nodes that share memory, we consider $\mathbb{M}_n = \mathbb{V}_n = 0$, and we define $\mathbb{U}_n$ as the total unified memory capacity, which combines both RAM and VRAM of node n.

During the bootstrap execution of the workload $\mathcal{W}$, a telemetry daemon records a multi-variate time-series matrix of resource utilization $\mathbf{R}(t)$ at time t, periodically every period θ. We denote by $\gamma = \lfloor \frac{T_{prof}}{\theta} \rfloor$ the number of periods used in the bootstrap phase.

$$\begin{aligned} \mathbf{R}(t) = [c(t), g(t), m(t), v(t), u(t)]^\top, \\ t \in [\theta, 2\theta, ..., \gamma\theta] \end{aligned} \tag{5}$$

We define an extrapolation function $\Phi : \mathbb{R}^{5 \times T_{prof}} \times (0, 1] \rightarrow \mathbb{R}^6$ mapping the bootstrap telemetry to the full-scale predicted workload footprint $\hat{\mathbf{\Omega}} = [\hat{T}_{total}, \hat{C}, \hat{G}, \hat{M}, \hat{V}, \hat{U}]$, see Figure 2. This prediction is the newly generated ground truth data that will be saved in the RAG database.

Memory consumption in deep learning workloads consists of static allocations (e.g., model weights, container context) and dynamic allocations (e.g., batch activations, gradients). Because the batch size B remains constant between the bootstrap phase and the full-scale execution, the peak memory utilization theoretically stabilizes once the first few batches are processed.

Consequently, we predict the full-scale peak memory by calculating the at full scale of the observed bootstrap memory footprint. However, to account for continuous memory fragmentation and potential memory leaks across long-running inferences, we introduce a monotonically decreasing safety margin $\delta(s)$.

For architectures with discrete memory pools, the predicted peak system RAM ($\hat{M}_{peak}$) and VRAM ($\hat{V}_{peak}$) are modeled as:

$$\hat{M}_{peak} = \sup_{t\in[\theta,\ldots,\gamma\theta]} m(t) + \delta_m(s) \tag{6}$$

$$\hat{V}_{peak} = \sup_{t\in[\theta,\ldots,\gamma\theta]} v(t) + \delta_v(s) \tag{7}$$

For unified memory architectures, both CPU and GPU processes compete for identical physical pages. The unified memory footprint $u(t)$ captures this shared allocation. The peak unified memory $\hat{U}_{peak}$ is extrapolated as:

$$\hat{U}_{peak} = \sup_{t\in[\theta,\ldots,\gamma\theta]} u(t) + \delta_u(s) \tag{8}$$

For the CPU utilization, similar to [12], we assume requests that are processed by the CPU are modeled by an $M/G/\infty$ queue. We denote $C(t)$ as the CPU utilization at time point t, $a_1, ..., a_E$ as the arrival times of E incoming events. Similarly, the departure time is defined as $d_1, ..., d_E$ with the processing times $p_1, ..., p_E$ so that $d_e = a_e + p_e$.

$$C(t) = \sum_{e=1}^{E} I(a_e < t < d_e) \tag{9}$$

with $I(X)$ being the indicator function, returning 1 if X is true, otherwise 0. The average CPU utilization $\tilde{C}(s\theta)$, used in Figure 2 as the input data for the ARIMA forecasting algorithm, can be computed with:

$$\begin{aligned}\tilde{C}(s\theta) &= \frac{1}{\theta}\int_{s\theta}^{(s+1)\theta} C(t)dt \\ &= \frac{1}{\theta}\int_{s\theta}^{(s+1)\theta} \sum_{e=1}^{E} I(a_e < t < d_e)dt \\ &= \frac{1}{\theta}\sum_{e=1}^{E}\int_{s\theta}^{(s+1)\theta} I(a_e < t < d_e)dt \\ &= \sum_{n=1}^{\theta}[F((s+1)\theta; a_e, d_e) - F(s\theta; a_e, d_e)]\end{aligned} \tag{10}$$

with

$$F(t; a_e, d_e) = \begin{cases} 0 & \text{if } t \le a_e \\ \frac{t-a_e}{\theta} & \text{if } a_e < t \le d_e \\ \frac{d_e-a_e}{\theta} & \text{if } t > d_e \end{cases} \tag{11}$$

We assume that we have received γ observations, $\tilde{C}(\theta), \tilde{C}(2\theta), ..., \tilde{C}(\gamma\theta)$. We aim at forecasting the future CPU utilization $\tilde{C}((\gamma+1)\theta), \tilde{C}((\gamma+2)\theta)$,... .

At time t, GPU utilization $G(t)$ is a function of GPU card parameters and the resource requirement of the AI workload [13]. Hence, potential occupancy limitations are imposed by resources such as registers, memory, and the number of streaming multiprocessors (SMs) required by the AI workload. Let $G(t)$ be the instantaneous GPU occupancy ratio, representing the fraction of the GPU thread capacity that is actively occupied, and given by:

$$\begin{aligned} G(t) &= \frac{\text{Active Thread per Block}}{\text{Thread per SM}} = \frac{T_b(t)}{T_{SM}}, \\ T_b(t) &= \begin{cases} B_b(t)\cdot Z_b(t) & \text{For thread block size} \\ \frac{R_{SM}}{R_b(t)} & \text{For register usage} \end{cases} \end{aligned} \tag{12}$$

where T_{SM} is the maximum number of resident threads per SM and $T_b(t)$ is the number of active resident threads induced by the running GPU kernel at time t, $B_b(t)$ is the number of active blocks per kernel, $Z_b(t)$ is the number of active threads per block, R_{SM} is the number of registers per SM, $R_b(t)$ is the number of registers required per block by the running kernel.

Similar to the CPU case, we define the average GPU utilization over the interval $[s\theta, (s+1)\theta]$ as

$$\tilde{G}(s\theta) = \frac{1}{\theta}\int_{s\theta}^{(s+1)\theta} G(t)dt. \tag{13}$$

Assume that L GPU kernel executions are observed within the measurement horizon. Let u_ℓ and v_ℓ denote the start and completion times of the ℓ-th GPU kernel, respectively, and let g_ℓ denote its corresponding occupancy level. Then, the instantaneous GPU utilization can be written as

$$G(t) = \sum_{\ell=1}^{L} g_\ell I(u_\ell < t < v_\ell), \tag{14}$$

where

$$g_\ell = \frac{1}{T_{SM}}\min\left\{B_{b,\ell}Z_{b,\ell}, \frac{R_{SM}}{R_{b,\ell}}, \right\}. \tag{15}$$

Substituting (14) into (13), the average GPU utilization is obtained as:

$$\begin{aligned}\tilde{G}(s\theta) &= \frac{1}{\theta}\int_{s\theta}^{(s+1)\theta}\sum_{\ell=1}^{L} g_\ell I(u_\ell < t < v_\ell)dt \\ &= \sum_{\ell=1}^{L} g_\ell\left[F_G((s+1)\theta; u_\ell, v_\ell) - F_G(s\theta; u_\ell, v_\ell)\right],\end{aligned} \tag{16}$$

where

$$F_G(t; u_\ell, v_\ell) = \begin{cases} 0, & t \le u_\ell, \\ \dfrac{t-u_\ell}{\theta}, & u_\ell < t \le v_\ell, \\ \dfrac{v_\ell - u_\ell}{\theta}, & t > v_\ell. \end{cases} \tag{17}$$

Therefore, $\tilde{G}(s\theta)$ quantifies the time-normalized GPU occupancy within the s-th control interval. We assume that we have received γ observations, $\tilde{G}(\theta), \tilde{G}(2\theta), \ldots, \tilde{G}(\gamma\theta)$, and we aim at forecasting the future GPU utilization $\tilde{G}((\gamma+1)\theta), \tilde{G}((\gamma+2)\theta), \ldots$.

A. ARIMA-Based Forecasting Model

Let's denote the discrete-time sequence of historical CPU or GPU utilization measurements as $X_t = \tilde{C}(t\theta)$ or $\tilde{G}(t\theta)$ for $t \in \{1, 2, \ldots, \gamma\}$. To employ the Auto-Regressive Integrated Moving Average (ARIMA) methodology [28], we must first stabilize the mean of the time series by applying a differencing operator of order d. Let B denote the backshift operator such that $BX_t = X_{t-1}$. The differenced, stationary series Y_t is obtained by:

$$Y_t = (1 - B)^d X_t \tag{18}$$

where $d \in \mathbb{Z}^+$ is the minimum integration order required to achieve stationarity.

Following the stabilization of the series, we model Y_t using an ARMA(p, q) process, which yields the comprehensive ARIMA(p, d, q)) model for the original sequence X_t. This captures the temporal correlation by expressing the current utilization as a linear combination of p past observations (autoregressive components) and q past forecast errors (moving average components), while the integration order d accounts for the number of prior differencing transformations required to render the raw, non-stationary data stationary. The comprehensive ARIMA(p, d, q) model for CPU utilization is thus formulated as:

$$\phi_p(B)(1 - B)^d X_t = \theta_q(B)\varepsilon_t \tag{19}$$

where $\varepsilon_t \sim \mathcal{WN}(0, \sigma^2)$ is a zero-mean white noise process representing stochastic, unpredictable fluctuations in CPU demand (e.g., microbursts). The autoregressive operator $\phi_p(B)$ and the moving average operator $\theta_q(B)$ are defined as characteristic polynomials of degrees p and q, respectively:

$$\phi_p(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p \tag{20}$$

$$\theta_q(B) = 1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q \tag{21}$$

To operationalize this model within the network controller, the structural hyperparameters (p, d, q) must be identified. This is achieved by iteratively evaluating candidate models and minimizing the Akaike Information Criterion (AIC). Once the optimal structure is selected, the parameter vectors $\mathbf{\Phi} = [\phi_1, \ldots, \phi_p]^\top$ and $\mathbf{\Theta} = [\theta_1, \ldots, \theta_q]^\top$ are estimated using Maximum Likelihood Estimation (MLE) over the historical observation window $[1, \gamma]$.

Finally, to reach a high precision forecast of the resource usage, we compute the k-step-ahead forecast of the CPU utilization, denoted as $\hat{X}_{\gamma+k} = \mathbb{E}[X_{\gamma+k} | X_1, \ldots, X_\gamma]$. Expanding the generalized difference equation, the one-step-ahead prediction for the imminent time epoch $(\gamma + 1)\theta$ is yielded by:

$$\hat{X}_{\gamma+1} = \sum_{i=1}^{p+d} \alpha_i X_{\gamma+1-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{\gamma+1-j} \tag{22}$$

where the coefficients α_i are derived from the algebraic expansion of the combined generalized autoregressive polynomial $\alpha(B) = \phi_p(B)(1 - B)^d$. This predictive formulation enables the orchestrator to continuously update $\hat{X}_{\gamma+1}$ as new telemetry data arrives. The same mathematical principles can be applied to the GPU, RAM and VRAM, or uRAM as well.

B. Adaptive Parameter Search Algorithm for Forecasting

Algorithm 1: Extended Cross-Validated Surrogate Search

Input: Matrix $\mathcal{M}$ of size $R \times C$, global limit T_{max}, Local threshold τ_{next}, Global threshold τ_{last}, Split ratio $\gamma \in (0, 0.8]$, Strategy Index $I \in \{1, 2, 3\}$

Output: Final estimated resource $\hat{\Omega}_{R,C}$

$H \leftarrow \emptyset$; $t_{total} \leftarrow 0$; $c_{skip} \leftarrow 0$; $F_{future} \leftarrow \emptyset$;
for $c \leftarrow 0$ **to** $C - 1$ **do**
  if $c_{skip} > 0$ **then**
    $c_{skip} \leftarrow c_{skip} - 1$; **continue**;
  $r_{skip} \leftarrow 0$;
  for $r \leftarrow 0$ **to** $R - 1$ **do**
    if $r_{skip} > 0$ **then**
      $r_{skip} \leftarrow r_{skip} - 1$; **continue**;
    $\Omega_{r,c}, t_{exe} \leftarrow$ ComputeResources($\mathcal{M}[r, c]$);
    $H \leftarrow H \cup \{\Omega_{r,c}\}$;
    $t_{total} \leftarrow t_{total} + t_{exe}$;
    if $t_{total} \geq T_{max}$ **then**
      return *TIMEOUT*, $F_{future}[|F_{future}| - 1].\hat{\Omega}$
    if $|H| < 8$ **then**
      continue ; // Mandatory Baseline
    else
      $status, \hat{\Omega}_{res}, L_c, L_r, F_{future} \leftarrow$ EvaluateSurrogateLeap $(H, \mathcal{M}, c, r, \Omega_{r,c}, \gamma, \tau_{last}, \tau_{next}, I, R)$;
      if $status =$ *SUCCESS* **then**
        return *SUCCESS*, $\hat{\Omega}_{res}$;
      if $status =$ *BREAK_COL* **then**
        $c_{skip} \leftarrow L_c$; **break**;
      if $status =$ *LEAP_ROW* **then**
        $r_{skip} \leftarrow L_r$;
return *EXHAUSTED*, $F_{future}[|F_{future}| - 1].\hat{\Omega}$;

To find the best possible parameter combination with which the ARIMA-based forecasting model is called, we implement an adaptive parameter search over the size of the training data and the number of epochs that the model will be trained on during the profiling phase, as depicted by the decision structure on the left of the *Active Profiling* module in Figure 2. These parameter combinations are denoted in matrix $\mathcal{M}$, where the columns represent the training samples $n_{samples}$ in steps of 1000 samples, and the rows represent the number of epochs n_{epochs}, always incremented by one. The start point $(0, 0)$ of the matrix resembles 1 epoch and 1000 training samples. The forecasting model is than initialized with $n_{samples}$ and n_{epochs} to predict the future resource usage $\hat{\Omega}$, but will as well return an accuracy score about the prediction, which we will use in the following to calculate the steps through the matrix $\mathcal{M}$ to find the best possible combination for a efficient forecast, while respecting the time constrains T_{max} of the request, to keep the forecasting in a reasonable time

Algorithm 2: Evaluate Surrogate and Leap Strategy

Input: $H, \mathcal{M}, c, r, \Omega_{r,c}, \gamma, \tau_{last}, \tau_{next}, I, R$
Output: Tuple $(status, \hat{\Omega}_{res}, L_c, L_r, F_{future})$

// Define Data Partitions
$v_{idx} \leftarrow \lfloor \gamma \times |H| \rfloor$;
$H_{train} \leftarrow H[0 : v_{idx} - 1]$;
$H_{val} \leftarrow H[v_{idx} : |H| - 1]$;
$U \leftarrow \{(x, y) \in \mathcal{M} \mid y > c \vee (y = c \wedge x > r)\}$;
$F_{val}, F_{future} \leftarrow$ COMPUTEARIMA(H_{train}, H_{val}, U);
// With $(acc_i, \hat{\Omega}_i) \leftarrow F[i]$
$acc_{next} \leftarrow F_{val}[0].acc$;
$\overline{acc} \leftarrow \frac{1}{|F_{val}|} \sum_{i=0}^{|F_{val}|-1} F_{val}[i].acc$;
if $\overline{acc} > \tau_{last}$ **then**
  if $U \neq \emptyset$ **then**
    return *(SUCCESS,* $F_{future}[|F_{future}| - 1].\hat{\Omega}, 0, 0, F_{future}$*)*;
  else
    return *(SUCCESS,* $\Omega_{r,c}, 0, 0, F_{future}$*)* ;
    // Matrix exhausted
if $acc_{next} > \tau_{next}$ **then**
  $skip\_col, L_c, L_r \leftarrow$ CALCLEAPDISTANCE(c, r, I, R);
  if $skip\_col$ **then**
    return *(BREAK_COL, NULL,* $L_c, 0, F_{future}$*)*;
  else
    return *(LEAP_ROW, NULL,* $0, L_r, F_{future}$*)*;
return *(CONTINUE, NULL,* $0, 0, F_{future}$*)*;

Algorithm 3: Calculate Multi-Strategy Leap

Input: Current col c, Current row r, Strategy Index I, Total rows R
Output: Tuple $(skip\_col, L_c, L_r)$ representing column abort, column leap, and row leap

if $I = 1$ **then**
  // Strategy 1: No leaping, exhaustive search
  return ***(false**, 0, 0)*;
else if $I = 2$ **then**
  // Strategy 2: Intra-column row leaping
  return ***(false**, 0, 3)*;
else if $I = 3$ **then**
  // Strategy 3: Inter-column tiered leaping
  $d \leftarrow (r + 1)/R$; // Calculate depth percentage
  if $d \leq \frac{1}{3}$ **then**
    return ***(true**, 3, 0)*
  else if $d \leq \frac{2}{3}$ **then**
    return ***(true**, 2, 0)*
  else
    return ***(true**, 1, 0)*
return ***(false**, 0, 0)* ; // Fallback safety condition

frame. To efficiently locate regions meeting the accuracy requirement within the matrix $\mathcal{M}$, we propose the Extended Cross-Validated Surrogate Search (ECVSS) (summarized in Algorithm 1). Unlike traditional exhaustive search methods that evaluate continuous sub-optimal regions, the proposed approach introduces a dynamic forward-leaping mechanism. By mathematically bypassing cells that can be predicted with a high accuracy, the algorithm rapidly converges on the target accuracy ϱ while strictly adhering to a computational time budget T_{max}.

To achieve this, ECVSS traverses $\mathcal{M}$ in a column-major sequence using an active learning loop. At its core, the algorithm operates on a "Compute-First" principle. For any evaluated spatial coordinate (r, c), the system mandatorily executes the exact resource computation to yield the ground-truth state $\Omega_{r,c}$, appending this value to a historical time-series vector H. To prevent predictive instability caused by a cold-start, the initial $n_{samples}$ coordinates are strictly computed without querying the forecasting engine, thereby establishing a robust empirical baseline.

Once the empirical baseline is established, ECVSS dynamically transitions into a rolling cross-validation paradigm (detailed in Algorithm 2). At each sequential coordinate step, the accumulated historical array H is partitioned into a training set H_{train} and a validation set H_{val} using a proportional split ratio γ. To mathematically guarantee the existence of validation data and prevent fatal division-by-zero anomalies during evaluation, this ratio is strictly bounded to the interval $\gamma \in (0, 0.8]$. Concurrently, the algorithm constructs a strictly forward-looking uncomputed queue $U \leftarrow \{(x, y) \in \mathcal{M} \mid y > c \vee (y = c \wedge x > r)\}$. The surrogate ARIMA model is trained exclusively on H_{train} to generate two distinct outputs: a scored predictive array over the validation set (F_{val}), which yields both estimated resources and empirical accuracy metrics, and an unscored macroscopic projection over the entire uncomputed future space (F_{future}).

By decoupling the scoring domain from the predictive domain, the algorithm rigorously evaluates model reliability on known ground-truth data without sacrificing its capacity for distant projection. The system calculates the mean accuracy of the validation forecasts, $\overline{acc}$. As shown in Figure 2, if $\overline{acc}$ exceeds the strict global threshold τ_{last}, it is assumed that the next predictions are done very accurately. At this point, the algorithm has enough confidence in the accuracy of the estimate for the resource usage to execute an early global termination. To eliminate predictive redundancy, the algorithm implements a semantic safeguard: if the matrix is fully exhausted ($U = \emptyset$), it yields the exact ground-truth state $\Omega_{r,c}$ with a SUCCESS flag. Otherwise, it projects to the absolute end of the uncomputed space, returning the final estimated resource $F_{future}[|F_{future}| - 1].\hat{\Omega}$, successfully resolving the search while bypassing all remaining computations.

If global convergence is not achieved, the subroutine eval-

uates the localized confidence of the immediate next validation point, defined as acc_{next}, to return a state control flag to the main traversal loop. An uncertain local forecast ($acc_{next} \leq \tau_{next}$) yields a `CONTINUE` flag, forcing the main algorithm to advance to the subsequent adjacent coordinate and perform an exact computation to iteratively reinforce H_{train} with new data. Conversely, a confident localized forecast ($acc_{next} > \tau_{next}$) queries the leaping logic (see Algorithm 3). We compare the performance of different leaping strategies. The magnitude and dimensional axis of this leap are dictated by a modular Strategy Index I. Depending on the selected index, the heuristic can return `CONTINUE` to enforce exhaustive verification ($I = 1$), return `LEAP_ROW` to execute an intra-column vertical leap to rapidly descend through the epoch parameter space ($I = 2$), or return `BREAK_COL` to perform an aggressive inter-column horizontal leap ($I = 3$). For $I = 3$, the horizontal magnitude is inversely proportional to the fractional matrix depth $d = (r+1)/R$, penalizing discoveries deep within the epoch rows to heavily incentivize early-column traversal.

Throughout the active search loop, the cumulative execution time t_{total} is continuously evaluated against the global constraint T_{max}. If the computational budget is exceeded, the algorithm triggers a `TIMEOUT` interrupt. To ensure graceful degradation, the algorithm avoids catastrophic failure by returning the furthest available projected resource state $F_{future}[|F_{future}| - 1].\hat{\Omega}$. Finally, if the traversal loop mathematically completes the entire configuration space without triggering global convergence or a timeout, the system outputs an `EXHAUSTED` flag alongside the final projection, ensuring strict semantic differentiation between a time-constrained interrupt and a fully traversed configuration search.

V. Data

To create the AI training data, we created 8 AI workloads that train on 6 different well known datasets. For the sake of comparative analysis, we distinguish between visual and tabular data and opt for classification algorithms only. The workloads ran on glsrpi, Thor Jetson and workstation PC, as examples of data for three different hardware options. The training was carried out with an increasing sample size and ran for 20 or 30 epochs, depending on the hardware. We chose the values with the highest epoch for the results shown. As can be seen in Table IV different datasets contain different amounts of data. Therefore, we chose to cap the Vision datasets to 30000 samples. The tabular datasets were capped at different lengths: The HIGGS dataset at 30000 rows, the forest coverage dataset at 10000 rows and the wine quality dataset at 4900 rows. This was necessary to prevent overfitting. The metrics measured include training time, testing time as well as GPU-, CPU-, RAM- and VRAM- or uRAM usage. While the training and testing time can be easily logged, for the GPU, CPU and different memory, we sampled the usage values every 10ms, which we estimate does not not put additional stress on the CPU. The data generated is then used to calculate the mean usage as well as the peak usage for the three metrics and is written in the dataset. This dataset also contains the measured testing accuracy as well as the testing loss. In the following

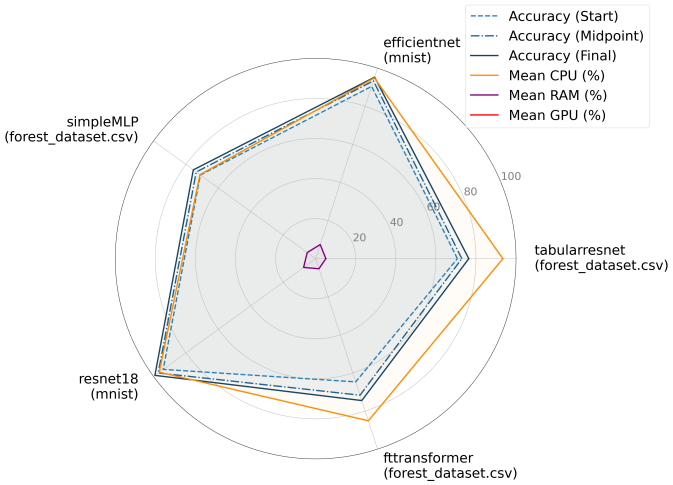


Figure 3: Accuracy, CPU usage and RAM usage for model training on the RPI. The combinations of model and dataset are shown on the outside, with the model name above and the dataset name in brackets. Tabular workloads can be distinguished by the addition of the CSV file type. As the device has no GPU, no GPU performance is shown.

we show some of the results from our least powerful and our most powerful device which also validate our approach. The full datasets are openly available and can be found [29].

A. RPI

Dataset generation was conducted using a Raspberry Pi (RPI) 5 equipped with 16 GB of RAM, representing the highest computational capacity currently available in this hardware series. Despite these specifications, the device remains resource-constrained for large-scale AI training. Consequently, the experimental scope was restricted to lightweight models with minimal parameter counts and limited to two datasets to mitigate onboard storage limitations. Furthermore, model training was capped at a maximum of 20 epochs, as extended processing times rendered the system impractical for real-world deployment. Figure 3 shows the testing accuracy of the first and last epoch as well as the middle epoch used, as well as the mean CPU, RAM, GPU and VRAM. Since the RPI does not have a GPU, the values for GPU and VRAM usage are always zero. What can be observed from this image, is that the main training bottleneck on the RPI is the CPU. Training vision AI workloads on CPU only, is possible, but time consuming as can be seen in Figures 4 and 5 which shows the training time per epoch over the increasing sample size, each for the maximum epochs of 20. The figures show the dependencies between the visual and tabular models, with the highest training time per epoch reaching over 10 minutes for ResNet18 and the lowest being less than one second for simpleMLP.

Overall, Figure 3 to Figure 5 show that the RPI is capable of training vision and tabular data feed for AI models. However, the training process is very time and CPU consuming especially for the vision tasks.

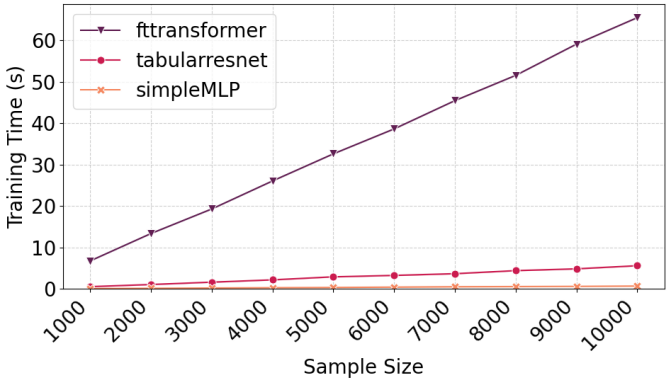


Figure 4: Training time per epoch over sample size for the tabular workload, for each of the tested models on the forest dataset, for 20 epochs on the RPI.

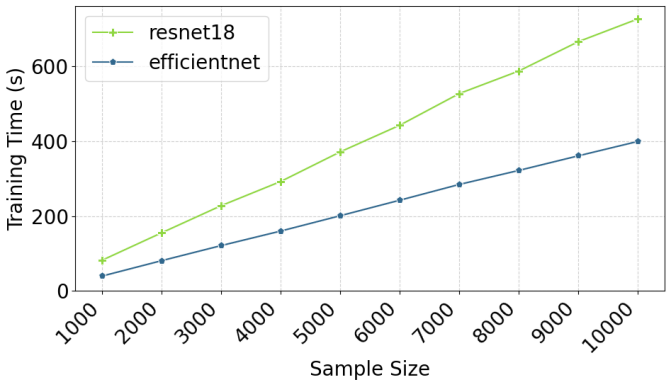


Figure 5: Training time per epoch over sample size for the vision workload, for each of the tested models on the MNIST dataset, for 20 epochs on the RPI.

B. JETSON

NVIDIA Jetson Thor, equipped with an NVIDIA Blackwell architecture GPU and 128 GB of unified memory (uRAM), has an enhanced computational capacity for experimentation, encompassing eight distinct models evaluated across three datasets each, as detailed in Tables IV and V. For these experiments, training durations were extended to a maximum of 30 epochs. Figure 6 illustrates the accuracy achieved at the initial, middle, and final epochs for all model-dataset configurations, alongside their corresponding mean GPU-, CPU-, and uRAM utilization. The results underscore the importance of optimizing training parameters; certain architectures exhibited near-optimal performance after a single epoch and showed marginal gains with prolonged training. For instance, MLP-Mixer demonstrated rapid convergence on MNIST dataset, and similarly showed no significant performance disparity between 15 and 30 epochs on CIFAR-10 dataset. Furthermore, the empirical data highlights divergence in GPU utilizations: vision-based models required two to three times more of the GPU resources than tabular models, whereas CPU and uRAM utilization remained highly consistent in all experimental combinations. The observed uRAM usage presents the entire observed system usage and not only the neural network footprint. By refining our measurement methodology to isolate process-specific memory and PyTorch's internal allocator, we successfully bypassed these system-level artifacts and observed that AI workloads allocated between 0.8% to 2.0% of the Jetson Thor's uRAM. Our data reflects the actual hardware behavior, which allocates 50% uRAM accurately describing the true operational baseline. Because all training and inference computations were explicitly delegated to GPU, CPU was primarily relegated to data loading and parameter management. Figure 7 and Figure 8 show the training time per epoch as the sample size increases on the Jetson platform for the eight evaluated tabular (Forest dataset) and visual (MNIST dataset) workloads. In Figure 7, the sample size is capped to 10000, which is the total cardinality of the dataset. Increasing this limit through oversampling would lead to severe model overfitting. Comparing these results with those presented in Figure 4 and Figure 5 shows that the Jetson provides substantial computational acceleration, processing visual and tabular workloads 10 and 20 times faster than the previously tested hardware, respectively. Finally, Figure 9 summarizes the mean per-epoch training latency across all evaluated model-dataset combinations at a fixed sample size of 10000. For the visual dataset, it shows that the choice of dataset has very little impact on mean training time. The same is true for the tabular data, with the wine quality dataset being the only outlier. This is because it is the smallest dataset, with the average being calculated for 5000 samples.

VI. Evaluation

This section describes the experimental evaluation of the AI Workload Agent system, comparing the full agentic architecture against a zero-shot LLM baseline.

A. Experimental Setup

Two prediction approaches are evaluated and compared against ground truth data: *1.) Zero-Shot LLM Baseline.* A simplified pipeline that routes all workloads exclusively through the LLM Zero-Shot Estimation node, bypassing the orchestrator, k-NN search, Active Profiling, and RAG components. This establishes a lower-bound baseline representing pure LLM inference without any system-level augmentation. *2.) Full Agentic Architecture.* The complete pipeline as described in Section III, comprising: A2A Interface; Policy Manager; k-NN Similarity Search on the profiling knowledge base; Reasoning Orchestrator (which routes to LLM Zero-Shot, Active Profiling, or LLM+RAG based on k-NN match confidence); Constraints Verification; and Re-Profile and Calibrate on policy violations. All predictions are evaluated against pre-computed ARIMA telemetry data containing 53 entries with measured values for training time, CPU usage, GPU usage, RAM usage and VRAM usage or unified RAM usage. All ARIMA strategies post an improvement over the full monitoring method, as the full monitoring would results in much higher latencies for the full assessment of the hardware values. With the ARIMA strategies we reach very comparable forecasts to the real world resource usage (see Figure 10a, 10b and 10c), but while the true monitoring time of the training

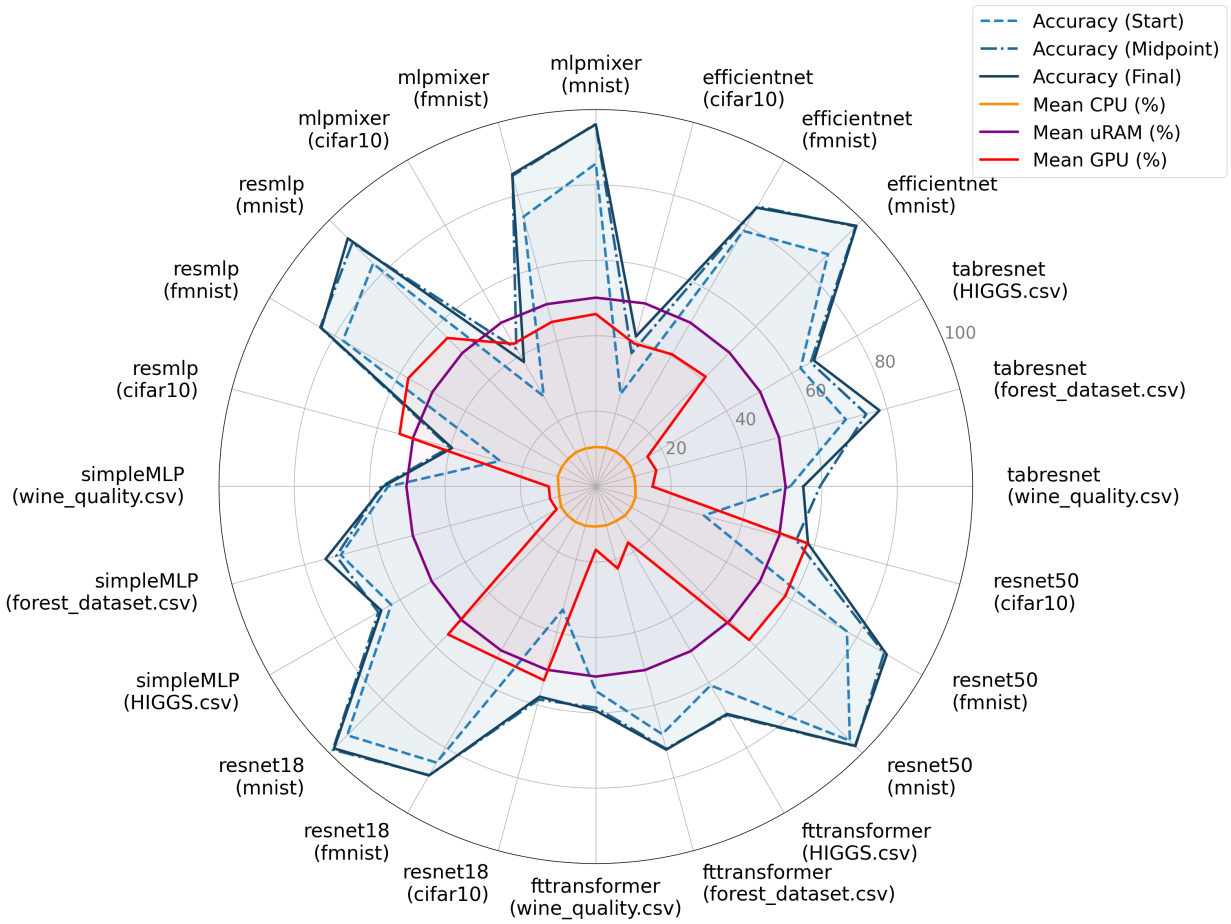


Figure 6: Testing Accuracy, Mean CPU, GPU and uRAM usage for the tested visual and tabular workloads on the JETSON Thor. The combinations of model and dataset are shown on the outside, with the model name above and the dataset name in brackets. Tabular workloads can be distinguished by the addition of the CSV file type. The uRAM and CPU values are constant for all combination, since training is carried out using the JETSONs GPU.

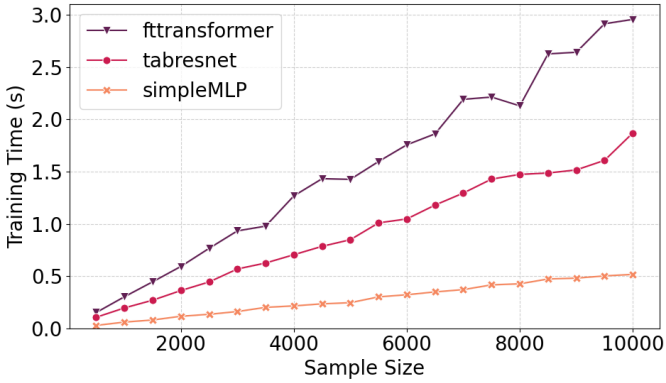


Figure 7: Training time per epoch over sample size for the forest dataset on the JETSON Thor, running for 30 epochs.

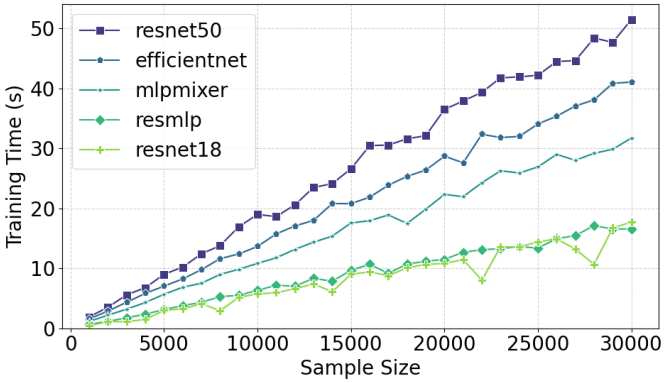


Figure 8: Training time per epoch over sample size for the MNIST dataset on the JETSON Thor, running for 30 epochs.

task for the Cifar10 dataset to train an EfficientNet on a Jetson Thor would take 1400 seconds the forecast times are between 425 to 890 seconds (see Figure 10d). We proved that it achieves comparable performance to Strategy 1 and the real world telemetry data, but with significantly less time. Averaged over all ARIMA forecasts Strategy 1 needs 693.22 seconds, Strategy 2 needs 473.61 seconds and Strategy 3 only takes 166.97 seconds. So for the ARIMA forecasting in the Full Agentic Architecture we are only using strategy 3.

All LLM inference is performed using the `mistralai/Magistral-Small-2509` model. This is a 7B parameter model optimized for instruction following

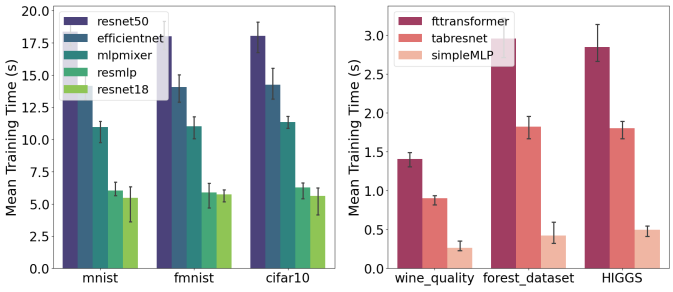
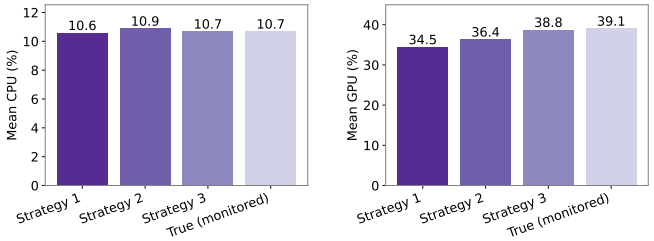
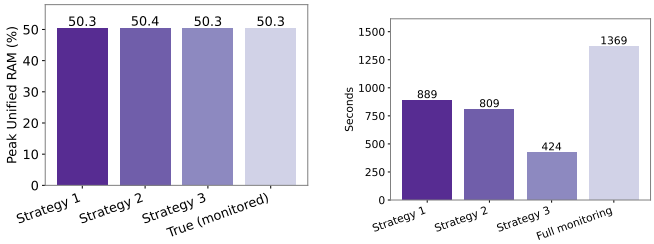


Figure 9: Mean training times per epoch for the visual and tabular datasets on the JETSON, showcasing all data and model combinations. The choice of data does not have a major influence on the mean training time per epoch.



(a) Forecasted vs actual CPU values

(b) Forecasted vs actual GPU values



(c) Forecasted vs actual uRAM values

(d) Time to successful forecast, full monitoring vs ARIMA

Figure 10: Corresponding ARIMA values for CPU, GPU, uRAM, and Execution Time for EfficientNet, cifar10 on the Jetson Thor.

and tool-use, making it suitable for the agentic pipeline's structured JSON output requirements.

B. Parameter Configuration

Table II lists all tunable parameters with their values and justifications. These settings are derived from the system's configuration files and agent implementations.

C. Workload Composition

53 evaluation workloads are tested and benchmarked. Six datasets are used: CIFAR-10, FMNIST, MNIST, Forest Covertype, HIGGS, and Wine Quality (all pre-processed as per the dataset preparation pipeline). The sample and epoch counts are fixed for each device in order to reflect realistic edge constraints. These differ from device to device so that each device operates at full capacity while still being able to complete its task. JetsonThor is tracking unified memory, not RAM and VRAM. RPI has no GPU (VRAM always 0.0%). PC with GPU uses dedicated VRAM (with a total VRAM of 4,080 MB). Due to the constrained nature of the RPI we could not benchmark all combinations on that hardware.

D. Evaluation Metrics

For each prediction, Mean Absolute Percentage Error (MAPE) is computed as:

$$\text{MAPE} = \frac{|\hat{y} - y|}{|y|} \times 100\% \tag{23}$$

where $\hat{y}$ is the predicted value and y is the ground truth value. MAPE is calculated per metric (Forecasting Time, CPU, GPU, RAM, VRAM, uRAM) and then averaged across all workloads for a given algorithm-dataset-device combination in the heatmaps which are here displayed for the RPI, the PC with GPU and the Nvidia Jetson Thor. Additionally all single combinations are plotted per data point in the scatter plots, where we combine the results for RAM and uRAM in one plot for increased comparability.

E. Results

Figure 11, 12, and 13 present the overall MAPE across all predicted metrics for workloads executed on the RPI, the PC with a GPU, and the NVIDIA Jetson Thor. The heatmaps compare the Zero-Shot LLM forecast against the proposed agentic framework, both with and without the inclusion of training time predictions.

As shown in Figure 11, the Zero-Shot LLM approach yields highly inaccurate predictions, with MAPE values broadly exceeding 200% and extreme outliers surpassing 4000%. Across all devices, predictions for vision algorithms and datasets exhibit comparatively lower errors than tabular datasets and algorithms, though the absolute error remains prohibitively high. Figure 12 demonstrates that the agentic system significantly improves forecasting accuracy. In optimal cases, the MAPE is reduced to approximately 0.4%. However, a few outliers persist within this framework across various hardware-algorithm combinations.Figure 13 displays the agentic framework's accuracy when training time predictions are excluded from the overall metric calculation. Under this condition, MAPE drops to near 0% for the vast majority of combinations. The few remaining outliers in this scenario correspond exclusively to edge cases where the system's knowledge base lacked any related workload data.

Figure 14 visualizes the predicted versus actual values for total training time, mean CPU usage, mean GPU usage, peak RAM, and peak VRAM, alongside the distribution of forecasting times. To maintain visibility in Figure 14a, extreme outliers have been filtered out. For the remaining data points, the Agentic framework and Zero-Shot LLM show comparable spread around the actual training times, though the Agentic framework accounts for the broader MAPE improvements noted previously. The divergence in accuracy becomes clearer in hardware utilization metrics. In Figure 14b, the Agentic

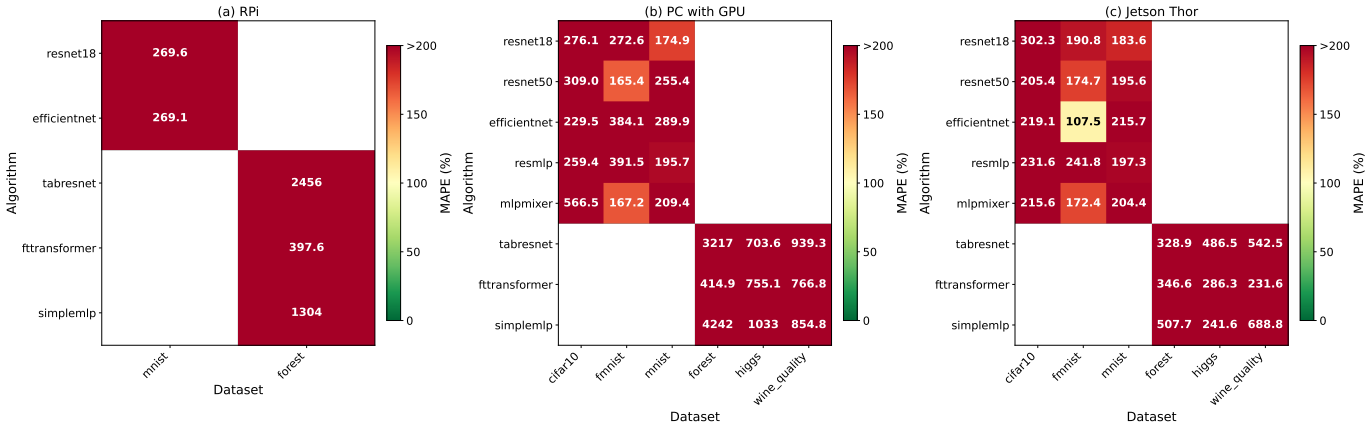


Figure 11: Overall prediction accuracy heatmap for the Zero-Shot LLM forecast

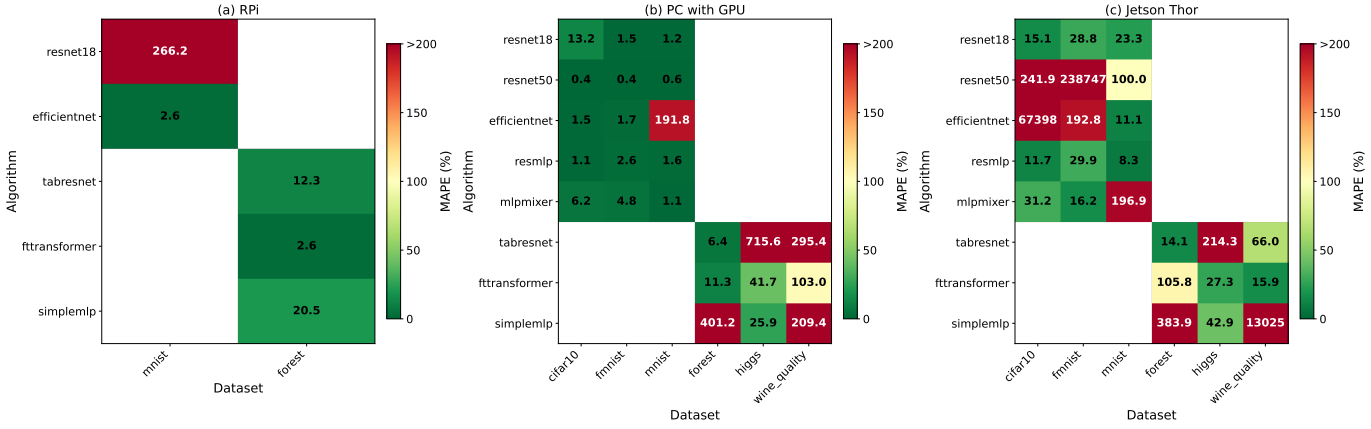


Figure 12: Overall prediction accuracy heatmap for the agentic framework

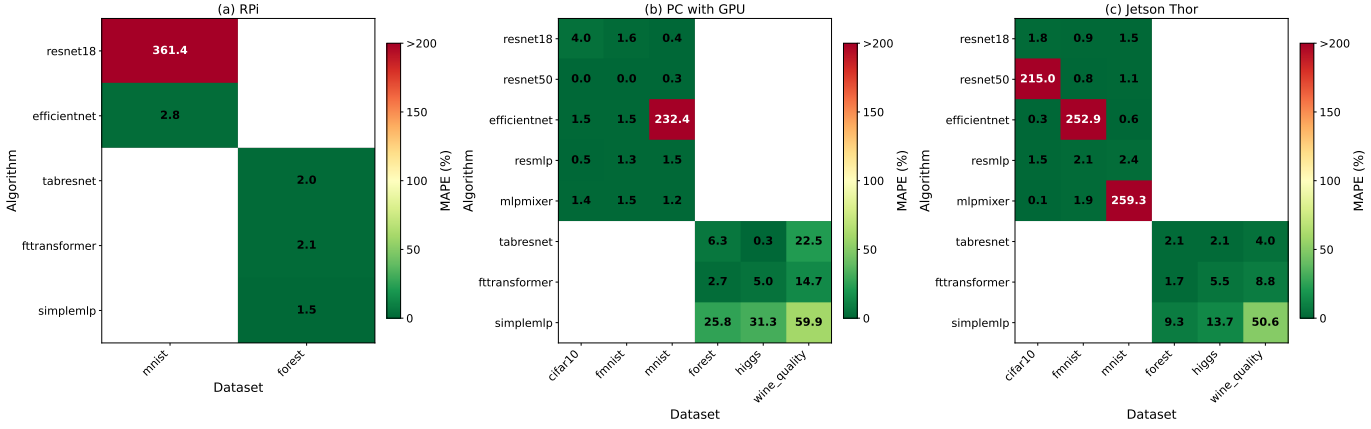


Figure 13: Overall prediction accuracy heatmap for the agentic framework without the training time prediction

Table II: AI Workload Agent Parameter Configuration

Category	Parameter	Value / Justification
LLM	Model	`mistralai/Magistral-Small-2509` (7B parameters, selected for instruction following and tool-use capability)
	Provider	TU Braunschweig KI-Toolbox API (local deployment, ensuring reproducibility and data privacy)
	Max Retries	3 (schema validation attempts; balances recovery from format errors against API latency)
RAG	Embedding model	`BAAI/bge-m3` (multilingual dense retrieval, selected for its strong performance on technical text similarity)
	Vector store	ChromaDB (in-memory, lightweight, sufficient for the growing KB of <100 entries)
	Chunk size / overlap	1000 / 200 (standard settings for JSON-structured telemetry documents, preserving field boundaries)
	Top-k retrieval	3 (limits retrieved context to the most relevant historical profiles, reducing token consumption)
k-NN Search	k (neighbors)	3 (small value appropriate for a knowledge base that grows from 0 to 53 entries during the experiment)
	Distance metric	Manhattan (L1 norm, robust to categorical feature mismatches and scale-invariant after preprocessing)
	Threshold mode	`closest` (only the nearest neighbor's distance is checked against the threshold, most conservative match)
	Distance threshold	2.0 (empirically tuned; balances false positives (overly broad matches) against false negatives (missing valid RAG candidates))
	Min matches required	2 (ensures at least two neighbors exist before declaring a threshold match, preventing single-outlier reliance)
	Categorical features	Algorithm, Dataset, Device (one-hot encoded)
	Numerical features	Samples, Epochs)
Policy	Max profiling duration	60 s (hard upper bound for Active Profiling telemetry retrieval; triggers timeout fallback to Zero-Shot)
	Max total forecasting time	600 s (absolute ceiling for the entire prediction pipeline, including retries and re-profiling)
Re-Profile	Max re-profile iterations	3 (prevents infinite loops on persistent constraint violations; after 3 failures, the pipeline returns the best-effort prediction)
Batch	Workloads	53 (complete set, covering all algorithm-dataset-device combinations)
	Execution order	Random (ensures unbiased KB population order; critical because RAG accuracy depends on prior profiling history)
	KB lifecycle	Empty at start → grows with each Active Profiling execution → reset after batch completion (guarantees reproducibility across runs)
Ground Truth	Monitored Hardware	24 combinations for NVIDIA Jetson, 24 combinations for GPU desktop PC, 5 combinations for RPi5

forecast closely tracks the perfect prediction line for mean CPU usage, whereas the Zero-Shot LLM consistently overestimates. A similar but less pronounced trend is visible for GPU usage (Figure 14c), where the Agentic framework exhibits lower variance than the Zero-Shot method, albeit with a slight tendency to underestimate actual usage. For memory metrics, the Agentic framework achieves near-perfect alignment with actual peak RAM (Figure 14d) and peak VRAM (Figure 14e), while the Zero-Shot LLM generates scattered and significantly inflated predictions. Figure 14f details the forecasting time required by each method. The Zero-Shot LLM is the fastest, averaging approximately 5.1 seconds. The Agentic framework requires slightly more time, averaging roughly 5.5 seconds. Both LLM-based approaches execute significantly faster and with vastly lower variance than the classical ARIMA baseline, which exhibits forecasting times ranging from tens of seconds up to over 600 seconds.

To evaluate the robustness of the Agentic framework, various hyperparameter and initial condition permutations were tested. Modifying the random seed for workload profiling yielded highly consistent outcomes, demonstrating algorithmic stability. However, the order of workload profiling measurably impacted both the knowledge base growth and prediction accuracy. Sorting workloads exclusively by device or by algorithm resulted in a smaller final knowledge base and the highest error rates for training time predictions.

Additionally, we evaluated the impact of the k-NN distance threshold on the RAG routing logic. A low similarity threshold frequently bypassed the RAG system entirely, as workloads rarely met the strict similarity criteria. In contrast, an excessively high threshold allowed unrelated workloads to trigger the RAG pipeline, preventing the knowledge base from expanding with new ground truth data.

F. Discussion

Our study reveals some limitations when using unaugmented LLMs for hardware-specific performance forecasting. The inaccuracy of the Zero-Shot LLM underscores that while LLMs possess broad conceptual knowledge of neural network architectures, they lack the intrinsic quantitative reasoning required to accurately estimate hardware-bound metrics without contextual grounding. It should be noted that our framework successfully addresses this issue by enriching the LLM with non-parametric memory via the RAG technique. By grounding the LLM's predictions in empirical data, the system successfully infers static or highly correlated metrics (such as memory footprint or parameter scaling) with near-perfect accuracy.

The results also reveal that training time prediction is still a critical bottleneck for the agentic system. Execution time is a highly dynamic metric, sensitive to non-linear hardware-specific factors that an LLM struggles to extrapolate. Variables such as thermal throttling on edge devices (evident on the Jetson Thor and RPI), memory bandwidth saturation, and I/O overhead do not scale linearly with dataset size or model parameters. Consequently, even when provided with closely

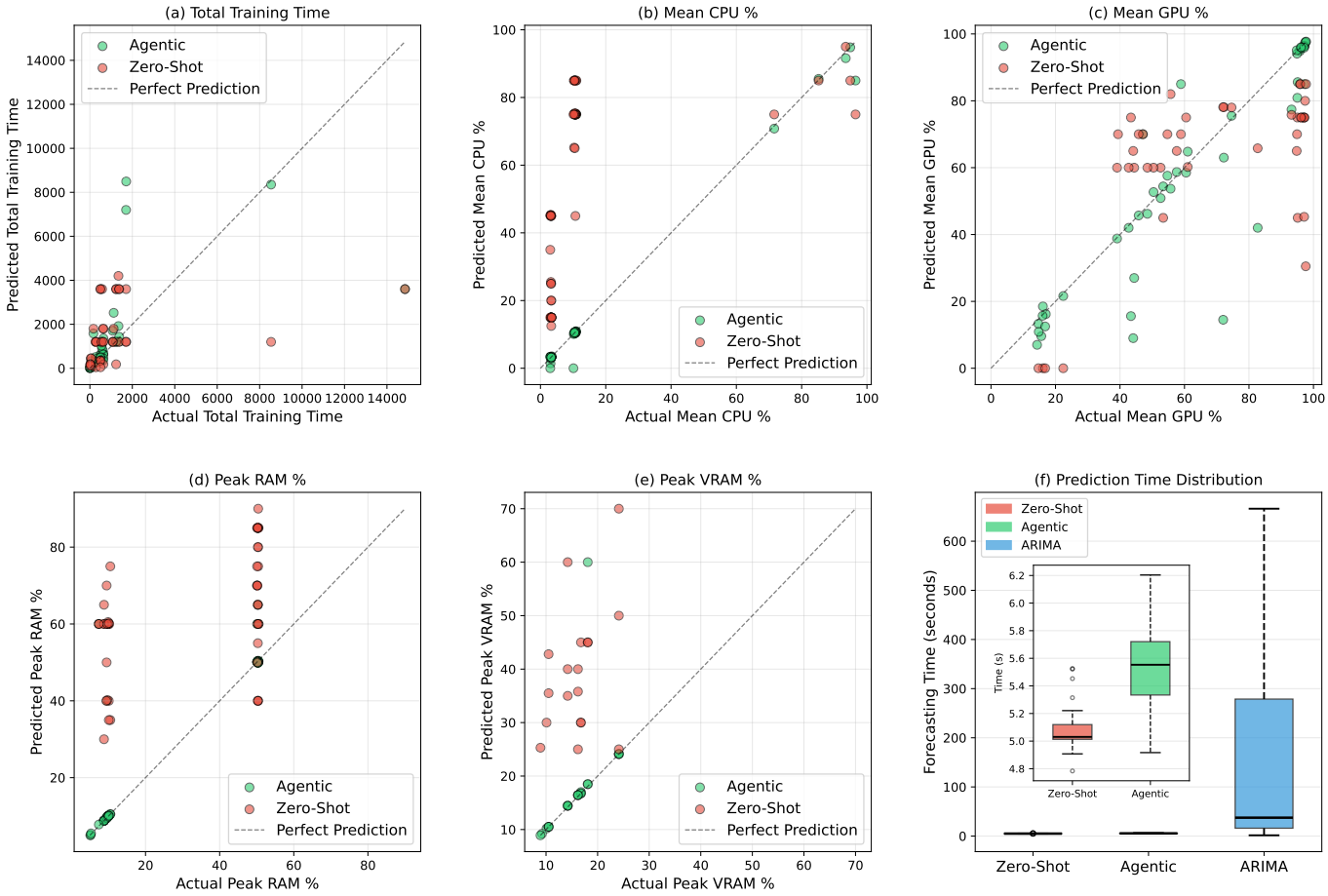


Figure 14: Predicted vs. Actual scatter plots for performance metrics (a-e) and Prediction Time Distribution (f).

Table III: Hardware and System Specifications

System / Device	CPU	GPU	RAM	Operating System
Raspberry Pi 5	Broadcom BCM2712 (Quad-core)	VideoCore VII	16GB	Ubuntu Server 25.04
NVIDIA Jetson Thor	NVIDIA Grace (Arm Neoverse V2)	NVIDIA Blackwell	128GB[a]	Jetson Linux
Custom Workstation	Intel i9-14900 (32) @ 5.500GHz	AMD Radeon RX 6400	32GB	NIXOS 24

[a]Unified Memory

Table IV: Dataset Specifications

Algorithm	Dataset Name	Dataset type	Num Rows	Num Columns	Num Classes	Num Images	Image Size (Pixels)	Document Type	Source
CNN	fashionMNIST	Images	-	-	10	70000	28x28(greyscale)	idx3-ubyte	[30]
	MNIST		-	-	10	70000	28x28 (greyscale)	idx3-ubyte	[31]
	CIFAR-10		-	-	10	60000	32x32 (RGB)	binary	[32]
MLP	wine_quality	Numerical	4900	12	10	-		csv	[33]
	HIGGS		> 1 Mio	29	2	-		csv	[34]
	forest_dataset		10.000	55	7	-		csv	[35]

Table V: Overview of Selected AI Architectures

Algorithm	Architecture Type	Approx. Parameters
ResNet-18	CNN	~11 Million
ResNet-50	CNN	~25.6 Million
EfficientNet (B0)	CNN	~5.3 Million
Simple MLP	Multi-Layer Perceptron	Variable
ResMLP (S12)	pure MLP	~15 Million
MLP-Mixer (B/16)	pure MLP	~59 Million
Tabular Resnet	Residual MLP (Tabular)	Variable (~0.1M – 2M)
FTTransformer	Attention-based (Tabular)	Variable (~1M – 5M)

related workloads via RAG, the LLM occasionally miscalculates temporal scaling factors, resulting in massive outliers.

Another limitation we observed in the experiments is that the system was susceptible to ”cold start” scenarios. When RAG cannot retrieve knowledge about a specific domain or dataset, the agentic framework inevitably regresses toward Zero-Shot performance levels, explaining the minor isolated outliers that persist. This specific limitation can be addressed by adopting a hybrid architecture: offloading execution time and latency estimations to a dedicated deterministic regressor

or an analytical hardware model, while preserving the agentic forecaster for resource prediction.

We also observe that the Agentic framework outperforms Zero-Shot LLM across all resource utilization metrics. This highlights the core advantage of the proposed architecture: the ability to gather new information and self-calibrate based on empirical ground truth. By passing similar workloads into the LLM's context window via RAG, the system anchors the LLM's inherently abstract hardware knowledge to factual, localized data. Most significantly, the accuracy does not come at the cost of operational delays. While the Agentic framework requires marginally more time than the Zero-Shot method due to the overhead of sequential LLM calls and database queries, it remains orders of magnitude faster than full monitoring or classical ARIMA forecasting.

Despite the operational advantages of our agentic framework, we also acknowledge several architectural and experimental limitations. First, the full integration of a secure sandbox environment is exceptionally complex. As a result, the sandbox mechanism in this study is emulated by using the created and described dataset. Deploying a fully functional, isolated execution environment on constrained edge devices introduces significant system-level integration challenges that fall outside the scope of this study. Second, the reliance on active monitoring and telemetry inherently introduces computational noise into the collected ground-truth data. Although eliminating this interference entirely is not possible in real-world edge deployments, we mitigated its impact by carefully calibrating the frequency of monitoring calls to get an optimal balance between data precision and profiling overhead.

In addition, the proposed system is currently agnostic to workload hyperparameters. Throughout our evaluation, we utilized a fixed set of hyperparameters to enable direct, equitable comparisons of AI workloads across heterogeneous hardware platforms. Exploring the vast space of possible hyperparameter combinations is computationally prohibitive and time-consuming, and doing so would ultimately prevent consistent, baseline comparisons of the underlying hardware performance. Furthermore, the dataset in this study has not been subjected to comprehensive statistical evaluation, which is a challenge. Lastly, our modeling of CPU and GPU is rather simplified. Modern heterogeneous processors utilize highly complex microarchitecture features, such as dynamic voltage and frequency scaling and proprietary scheduling algorithms, which needs to be addressed in future work.

VII. Conclusion

We proposed and evaluated a novel autonomous agentic framework designed to profile and allocate resources for zero-knowledge AI workloads in highly heterogeneous and resource-constrained edge computing environments. By integrating a continuous self-calibration mechanism, our approach successfully addresses the fundamental challenges of operational drift and the absence of reliable ground truth typically encountered in open-ended, decentralized deployments.

The main contribution of our study is the self-calibrating AI Workload Agent capable of orchestrating four complementary profiling modules. We demonstrated that while zero-shot LLM estimation suffers from significant prediction errors (exceeding 200% MAPE) when evaluating unknown executables, combining hardware resource monitoring with ARIMA-based forecasting establishes a highly reliable ground truth. Crucially, our novel adaptive parameter search utilizing three leaping strategies allows the system to dynamically approximate full-scale resource footprints and terminate early. Our comprehensive evaluation on a new open benchmark dataset across diverse hardware platforms (Raspberry Pi 5, NVIDIA Jetson Thor, and a GPU workstation) validates the framework's efficacy. By synthesizing empirical telemetry with a RAG-enhanced estimation module, the system successfully refines its own knowledge base, reducing prediction errors to single-digit MAPE for well-covered workload classes. Furthermore, the proposed ARIMA leaping algorithm proved 52% faster than classical models while maintaining equivalent accuracy.

Ultimately, this self-calibrating architecture provides a critical foundation for deploying reliable agentic edge intelligence. To further mature this framework, future research must address several key operational and architectural challenges. First, we will investigate comparing the ARIMA-based module with lightweight, on-device sequential neural networks (such as LSTMs or state-space models) to better capture non-linear resource degradation and transient microbursts. Because forecasting training time remains a significant bottleneck for agentic architectures, we also plan to integrate dedicated neural network-based runtime predictors. Furthermore, the system can be improved by optimizing the initial seeding strategy and implement dynamic, self adjusting k-NN thresholds to ensure robust RAG retrieval across varying edge computing environments. To complement this and to ensure long-term scalability on constrained edge orchestrators, future iterations must implement dynamic memory management and eviction policies to prevent unbounded RAG knowledge base growth. We propose to extend the single-agent orchestrator into a fully decentralized, multi-agent cooperative system.

References

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, 2019.

[2] L. Lovén, R. Farahani, I. Murturi, S. Sigg, and S. Dustdar, "Agentic edge intelligence: A research agenda," in *Proceedings of the 18th IEEE/ACM International Conference on Utility and Cloud Computing*. New York, NY, USA: Association for Computing Machinery, 2026.

[3] J. François, A. Clemm, D. Papadimitriou, S. Fernandes, and S. Schneider, "Research Challenges in Coupling Artificial Intelligence and Network Management," Internet Engineering Task Force, Internet-Draft draft-irtf-nmrg-ai-challenges-05, Mar. 2025, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-nmrg-ai-challenges/05/

[4] Y. Huang, H. Du, X. Zhang, D. Niyato, J. Kang, Z. Xiong, S. Wang, and T. Huang, "Large language models for networking: Applications, enabling techniques, and challenges," *IEEE Network*, vol. 39, no. 1, pp. 235–242, 2025.

[5] G. Bovenzi, F. Cerasuolo, D. Ciuonzo, D. Di Monda, I. Guarino, A. Montieri, V. Persico, and A. Pescapé, "Mapping the landscape of generative ai in network monitoring and management," *IEEE Transactions on Network and Service Management*, vol. 22, no. 3, pp. 2441–2472, 2025.

[6] M. Asif, T. Ahmed Khan, and W.-C. Song, "Evaluating large language models for optimized intent translation and contradiction detection using knn in ibn," *IEEE Access*, vol. 13, pp. 20 316–20 327, 2025.

[7] C. Provvedi, L. Seidenari, B. Picano, and R. Fantacci, "Intent-llm: A framework for automated network configuration through code generation," *IEEE Transactions on Cognitive Communications and Networking*, vol. 12, pp. 7246–7258, 2026.

[8] K. Dzeparoska and A. Leon-Garcia, "Kpi assurance and llms for intent-based management," in *NOMS 2025-2025 IEEE Network Operations and Management Symposium*, 2025, pp. 1–9.

[9] S. Taheri, A. Ihalage, P. Mishra, S. Coaker, F. Muhammad, and H. Al-Raweshidy, "Domain tailored large language models for log mask prediction in cellular network diagnostics," *IEEE Transactions on Network and Service Management*, vol. 22, no. 3, pp. 2370–2381, 2025.

[10] K. Dzeparoska, A. Tizghadam, and A. Leon-Garcia, "Emergence: An intent fulfillment system," *IEEE Communications Magazine*, vol. 62, no. 6, pp. 36–41, 2024.

[11] R. Zhang, S. Tang, Y. Liu, D. Niyato, Z. Xiong, S. Sun, S. Mao, and Z. Han, "Toward agentic AI: Generative information retrieval inspired intelligent communications and networking," *IEEE Communications Magazine*, vol. 64, no. 1, pp. 197–204, 2026.

[12] H. L. Hammer, A. Yazidi, A. Bratterud, H. Haugerud, and B. Feng, "A queue model for reliable forecasting of future cpu consumption," *Mobile Networks and Applications*, vol. 23, no. 4, pp. 840–853, 2018.

[13] M. R. Pimple and S. R. Sathe, "Analysis of resource utilization on gpu," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 2, 2019, copyright - © 2019. This work is licensed under https://creativecommons.org/licenses/by/4.0/ (the "License"). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Last updated - 2023-11-25. [Online]. Available: https://www.proquest.com/scholarly-journals/analysis-resource-utilization-on-gpu/docview/2656394679/se-2

[14] R. Marotta, G. Russo Russo, F. Quaglia, and P. Di Sanzo, "A bootstrapping technique for reducing the costs of machine learning models for predicting execution times in iaas clouds," in *Proceedings of the 2025 ACM Symposium on Cloud Computing*, ser. SoCC '25. New York, NY, USA: Association for Computing Machinery, 2026.

[15] Z. Yuan, X. Wang, Y. Nie, Y. Tao, Y. Li, Z. Shao, X. Liao, B. Li, and H. Jin, "Dynpipe: Toward dynamic end-to-end pipeline parallelism for interference-aware dnn training," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 11, pp. 2366–2382, 2025.

[16] H. Sfaxi, I. Lahyani, S. Yangui, and M. Torjmen, "Latency-aware and proactive service placement for edge computing," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 4243–4254, 2024.

[17] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using arima model and its impact on cloud applications' qos," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015.

[18] R. Singh, P. Sarkar, S. M. Turjya, J. Dansana, B. Agarwalla, and A. Bandyopadhyay, "Advanced deep learning techniques for workload forecasting in fog computing, elevating performance efficiency," in *2024 International Conference on Intelligent Computing and Emerging Communication Technologies (ICEC)*, 2024, pp. 1–6.

[19] M. Kreutzer, K. Dudzik, J. Wang, V. P. Betancourt, and J. Becker, "Execution time prediction via lightweight ai for edge-fog-cloud task offloading," in *2025 IEEE Annual Congress on Artificial Intelligence of Things (AIoT)*, 2025, pp. 286–293.

[20] L. Nkenyereye, C. Rajkumar, B. G. Lee, and W.-Y. Chung, "Dynamic transfer learning switching approach using resource benchmark in edge intelligence," *IEEE Internet of Things Journal*, vol. 12, no. 13, pp. 25 148–25 170, 2025.

[21] Y. Chen, Y. Sun, H. Yu, and T. Taleb, "Joint task and computing resource allocation in distributed edge computing systems via multi-agent deep reinforcement learning," *IEEE Transactions on Network Science and Engineering*, 2024.

[22] M. A. Hady, S. Hu, M. Pratama, Z. Cao, and R. Kowalczyk, "Multi-agent reinforcement learning for resources allocation optimization: a survey," *Artificial Intelligence Review*, vol. 58, no. 11, p. 354, 2025.

[23] G. Yao, H. Liu, and L. Dai, "Multi-agent reinforcement learning for adaptive resource orchestration in cloud-native clusters," in *Proceedings of the 2nd International Conference on Intelligent Computing and Data Analysis*, ser. ICDA '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 680–687.

[24] Q. Liu, J. Yang, and Z. Yan, "Dynamic resource orchestration in edge computing environments using multi-agent reinforcement learning," *Knowledge and Information Systems*, 2025.

[25] X. Chen, J. Cao, R. Cao, Y. Sahni, M. Zhang, and Y. Ji, "Decentralized task offloading in collaborative edge computing: A digital twin assisted multi-agent reinforcement learning approach," *IEEE Transactions on Mobile Computing*, vol. 25, no. 4, pp. 4776–4790, 2026.

[26] X. Wang, J. He, Z. Tang, J. Guo, J. Lou, L. Qian, T. Wang, and W. Jia, "Adaptive ai agent placement and migration in edge intelligence systems," in *2025 IEEE 25th International Conference on Communication Technology (ICCT)*, 2025.

[27] Y. Lin, S. Peng, Y. Li, S. Luo, H. Shen, K. Ye, and C. Xu, "Workload-adapted resource allocation for llm distributed serving in serverless clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 37, no. 6, pp. 1442–1457, 2026.

[28] K. Ibrahimi and Y. Serbouti, "Prediction of the content popularity in the 5g network: Auto-regressive, moving-average and exponential smoothing approaches," in *2017 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, 2017, pp. 1–7.

[29] F. Gentzen, M. Grunewald, I. Zacarias, M. Bensalem, and A. Jukan, "Edge_Device_AI_Hardware_monitore_Dataset," https://github.com/MarlaGru/Edge_Device_AI_Hardware_monitore_Dataset, 2024.

[30] Kaggle, "Fashion mnist dataset," https://www.kaggle.com/datasets/zalando-research/fashionmnist, 2026, accessed: 2026-07-03.

[31] ——, "Mnist dataset," https://www.kaggle.com/datasets/hojjatk/mnist-dataset, 2026, accessed: 2026-07-03.

[32] ——, "Cifar- 10 dataset," https://www.kaggle.com/c/cifar-10, 2026, accessed: 2026-07-03.

[33] ——, "Wine quality dataset," https://www.kaggle.com/datasets/yasserh/wine-quality-dataset, 2026, accessed: 2026-07-03.

[34] ——, "Higgs dataset," https://www.kaggle.com/datasets/erikbiswas/higgs-uci-dataset, 2026, accessed: 2026-07-03.

[35] ——, "Forest dataset," https://www.kaggle.com/datasets/uciml/forest-cover-type-dataset, 2026, accessed: 2026-07-03.