Proceedings of the International Conference on the Internet of Things (IoT) Workshops 2025

Pantelis Frangoudis / Dragi Kimovski (Eds.)

2025
Institute of Information Systems Engineering
TU Wien

Publisher

Institute of Information Systems Engineering, TU Wien (2025)
Favoritenstraße 9-11/194, 1040 Wien, Austria

Editors (responsible for the content)

Pantelis Frangoudis, TU Wien
Dragi Kimovski, University of Klagenfurt

ISBN: 978-3-200-10918-6
ISSN: 3061-1008
DOI: 10.34749/3061-1008.2025

Preface

It is our great pleasure to present you the **Proceedings of the Workshops held in conjunction with the 15th International Conference on the Internet of Things (IoT 2025)** on November 18, 2025, at Vienna University of Technology (TU Wien) in Vienna, Austria. Two specialized workshops were organized, focusing on different key aspects of contemporary IoT systems, in addition to a hands-on tutorial on orchestrating federated learning services. Following a peer review process by the organizers and technical program committee members, a number of papers were selected for presentation and are included in this volume. The program further included exciting keynote talks and panel sessions. The following events took place:

- **LongevIoT 2025: 2nd International Workshop on Longevity in IoT Systems**, organized by Boris Sedlak (TU Wien, Austria), Malte Josten (University of Duisburg-Essen, Germany), and Peter Zdankin (University of Duisburg-Essen, Germany). LongevIoT 2025 focused on challenges associated with the premature aging of IoT systems and the quest for IoT longevity.

- **ScaleSys 2025: 1st International Workshop on Intelligent and Scalable Systems across the Computing Continuum**, organized by Reza Farahani (University of Klagenfurt, Austria), Lorenzo Carnevale (University of Messina, Italy), Nishant Saurabh (Utrecht University, the Netherlands), and Gabriele Russo Russo (University of Rome Tor Vergata, Italy). ScaleSys 2025 addresses the intersection of computing systems and AI, particularly with respect to building and operating scalable, intelligent, and sustainable systems and services across the edge–cloud computing continuum.

- **Tutorial on Orchestrating Hierarchical Federated Learning Pipelines with the AIoTwin Middleware**, organized by Ivan Čilić, Ana Petra Jukić, Katarina Vuknić, and Ivana Podnar Žarko (University of Zagreb, Croatia) and including presentations and hands-on sessions.

We would like to thank the workshop and tutorial organizers, including the members of the technical program committees, for their efforts to create a very interesting program, and the authors for submitting their latest research findings. We extend our gratitude to the IoT 2025 general chairs, as well as the local organizers at TU Wien for their relentless efforts. Finally, we would

like to acknowledge the support of TU Wien Bibliothek in the preparation of this proceedings volume and for hosting it online.

We sincerely hope that you find the contributions included in this volume interesting and inspiring.

IoT 2025 Workshop Chairs

Pantelis Frangoudis, TU Wien
Dragi Kimovski, University of Klagenfurt

Committees

LongevIoT 2025: 2nd International Workshop on Longevity in IoT Systems

Organizers

- Boris Sedlak (TU Wien, Austria)
- Malte Josten (University of Duisburg-Essen, Germany)
- Peter Zdankin (University of Duisburg-Essen, Germany)

Technical Program Committee

- Chao Qian (University of Duisburg-Essen, Germany)
- Eirini E. Tsiropoulou (Arizona State University, USA)
- Jan S. Rellermeyer (University of Hannover, Germany)
- Koojana Kuladinithi (TU Hamburg, Germany)
- Lorenz Schwittmann (Independent Researcher)
- Marco Picone (Università di Modena e Reggio Emilia, Italy)
- Stephan Sigg (Aalto University, Finland)
- Suzan Bayhan (University of Twente, Netherlands)
- Torben Weis (University of Duisburg-Essen, Germany)
- Victor C. Pujol (Universität Pompeu Fabra, Spain)
- Waldir Moreira (Fraunhofer Portugal AICOS, Portugal)

ScaleSys 2025: 1st International Workshop on Intelligent and Scalable Systems across the Computing Continuum

Organizers

- Reza Farahani (University of Klagenfurt, Austria)
- Lorenzo Carnevale (University of Messina, Italy)
- Nishant Saurabh (Utrecht University, the Netherlands)

- Gabriele Russo Russo (University of Rome Tor Vergata, Italy)

Technical Program Committee

- Abdelhak Bentaleb (Concordia University, Canada)
- Adel N. Toosi (University of Melbourne, Australia)
- Arda Goknil (Sintef, Norway)
- Atakan Aral (University of Vienna, Austria)
- Auday Al-Dulaimy (Mälardalen University, Sweden)
- Carolina Fortuna (JSI, Slovenia)
- Daniel Balouek (Inria, France)
- Daniel Thilo Schroeder (Sintef, Norway)
- Dragi Kimovski (University of Klagenfurt, Austria)
- Frank Loh (University of Würzburg, Germany)
- Habib Mostafaei (Eindhoven University of Technology, Netherlands)
- Ilir Murturi (TU Vienna, Austria)
- Jože Martin Rožanec (University of Twente, Netherlands)
- José Santos (Ghent University, Belgium)
- Kuan-Hsun Chen (University of Twente, Netherlands)
- Kurt Horvath (University of Klagenfurt, Austria)
- Lauritz Thamsen (University of Glasgow, UK)
- Luiz Bittencourt (Unicamp, Brazil)
- Maria Fazio (University of Messina, Italy)
- Mohammad Shojafar (University of Surrey, United Kingdam)
- Naser Hossein Motlagh (University of Helsinki, Finland)
- Sagar San (Sintef, Norway)

- Shashikant Ilager (University of Amesterdam, Netherlands)
- Sören Henning (Dynatrace Research, Austria)
- Zhiming Zhao (University of Amesterdam, Netherlands)

Advisory Board

- Alexandru Iosup (VU University Amsterdam, Netherlands)
- Ana Lucia Varbanescu (University of Twente, Netherlands)
- Christian Timmerer (University of Klagenfurt, Austria)
- Radu Prodan (University of Innsbruck, Austria)

Tutorial on Orchestrating Hierarchical Federated Learning Pipelines with the AIoTwin Middleware

Organizers

- Ivan Čilić (University of Zagreb, Croatia)
- Ana Petra Jukić (University of Zagreb, Croatia)
- Katarina Vuknić (University of Zagreb, Croatia)
- Ivana Podnar Žarko (University of Zagreb, Croatia)

Contents

Neural Caching: Improving Longevity of Smart IoT Devices running Artificial Neural Networks

Christian Sallinger
christian.sallinger@tuwien.ac.at
TU Wien
Vienna, Austria

Christian Stippel
christian.stippel@tuwien.ac.at
TU Wien
Vienna, Austria

Elias Panner
elias.panner@student.tuwien.ac.at
TU Wien
Vienna, Austria

Paul Poschenreither
paul.poschenreither@fraunhofer.at
Fraunhofer Austria Research GmbH
Vienna, Austria

Ralph Hoch
ralph.hoch@vactory.at
Digital Factory Vorarlberg GmbH
Dornbirn, Austria

Benjamin Schwendinger
benjamin.schwendinger@fraunhofer.at
Fraunhofer Austria Research GmbH
Vienna, Austria

Abstract

Resource constraints and hardware aging make long-term neural inference on embedded and IoT devices increasingly challenging. We present *Neural Caching*, a lightweight inference mechanism that exploits the piecewise-linear structure of Artificial Neural Networks (ANNs) with piecewise linear activation functions to accelerate evaluation without retraining or model compression. Our approach, *Neural Caching*, associates each locally linear region of an ANN with a cached affine mapping and reuses it for subsequent inputs falling within the same or nearby activation region. This enables full predictions to be computed via a single matrix multiplication instead of potential multiple matrix multiplications. Experiments on four human-activity recognition datasets demonstrate up to an order-of-magnitude reduction in inference time while preserving classification performance metrics within ±0.1 % of baseline accuracy. By lowering computational load and power draw, neural caching extends device lifetime and supports sustainable, long-term deployment of machine-learning models in real-world IoT environments.

CCS Concepts

• **Computer systems organization** → **Neural networks**; • **Theory of computation** → *Caching and paging algorithms.*

Keywords

artificial neural networks, inference-time caching, hardware longevity, time-series classification

1 Introduction

The rapid proliferation of IoT devices has fundamentally changed how we sense, collect, and act upon environmental data. Millions of connected sensors now operate continuously in smart cities, industrial monitoring, agriculture, and healthcare, often in resource-constrained and physically exposed environments. A persistent challenge in these systems is *longevity*: as hardware ages, sensors and embedded processors experience gradual degradation—battery capacities decline, compute throughput diminishes due to transistor wear and thermal stress [Kraak et al. 2018; Zaidi et al. 2024], and replacement is often infeasible. Sustaining reliable and efficient machine-learning inference under such conditions remains an open problem.

Neural networks have become a dominant approach for analyzing complex sensor data, owing to their ability to capture nonlinear dependencies and generalize across heterogeneous and noisy environments. However, their deployment on IoT devices remains limited by computational and energy constraints: deep models require substantial processing power and memory, accelerating battery depletion and reducing device lifespan. Common model-compression techniques—such as pruning, quantization, and knowledge distillation—help mitigate these costs, yet they often require retraining and may introduce slight losses in predictive accuracy. Moreover, most of these approaches assume static hardware performance and do not adapt to gradual degradation or varying operating conditions over time.

Recent theoretical work has shown that neural networks with piecewise-linear activation functions (such as Rectified Linear Unit (ReLU) or hard tanh) partition their input space into exponentially many linear regions [Montúfar et al. 2014]. Within each region, the network behaves as a single affine transformation. This structure opens the door to alternative forms of acceleration that exploit redundancy in repeated inference: if multiple inputs fall into the same linear region, their outputs can be computed by a cached linear mapping instead of full forward propagation.

In this work, we build upon this insight and propose an inference-time caching mechanism tailored for IoT time-series models. While caching has been explored in convolutional and vision networks—e.g., through feature reuse across video frames or linear-region enumeration for small Multilayer Perceptrons (MLPs) [Joyce and Verschelde 2024, 2025]—it has not, to our knowledge, been applied to continuous time-series data on embedded IoT platforms. Our method exploits the local linearity of ReLU networks to cache region-wise affine transformations encountered during normal operation. Subsequent inferences that fall into known regions are executed by lightweight matrix multiplications, yielding substantial compute and energy savings.

This approach directly addresses the longevity issue of IoT devices. As compute resources degrade and batteries age, the algorithm maintains acceptable latency through increasing cache hit rates, effectively compensating for lost performance. In addition, by reducing active computation and the number of recharge cycles, it extends sensor lifetime and supports sustainable, long-term operation without sacrificing model capacity. Larger, more generalizable models—previously impractical on constrained hardware—can thus be deployed efficiently using our region-based caching scheme.

Our main contributions are as follows:

- We formulate a region-based caching algorithm for piecewise-linear neural networks and adapt it for time-series inference on embedded IoT devices.
- We demonstrate that caching affine mappings across linear regions yields consistent inference-time speedups as hardware performance degrades.
- We show that our approach preserves model accuracy while significantly reducing used computation power, thereby prolonging device longevity.

The remainder of this paper is organized as follows. Section 2 reviews related work on neural network acceleration and caching. Section 3 outlines the theoretical background of ReLU linear regions. Section 4 introduces the proposed Hierarchical Hypersphere Caching algorithm, followed by experiments and results in Section 5. Section 6 shows the limitations of our work. Section 7 concludes the paper and discusses future research directions.

2 Related Work

Neural-network acceleration without retraining has been explored from several complementary directions, including (i) exploiting piecewise-linear structure, (ii) inference-time caching, (iii) conditional computation, and (iv) lookup-table acceleration.

Piecewise-Linear and Region-Based Methods. ReLU networks partition input space into linear regions, each defining an affine mapping. Early *locally weighted learning* approaches such as locally weighted regression and locally weighted projection regression [Atkeson et al. 1997] approximated nonlinear functions through local linear models, effectively forming a piecewise-linear surface but requiring large memory and neighbor search—impractical for embedded devices. Mixture-of-Experts (MoE) networks [Shazeer et al. 2017] achieve conditional computation by activating only a subset of sub-networks per input, but increase model size and training complexity. Joyce and Verschelde [Joyce and Verschelde 2025] formally characterized how each activation pattern in a ReLU network corresponds to a unique affine mapping, providing the theoretical basis for region caching. Their work was primarily analytical and demonstrated on small multilayer perceptrons. Building on this principle, we extend the concept to time-series inference on IoT hardware, where activation patterns recur naturally. Unlike prior theoretical studies, we address limited cache memory, hardware degradation, and continuous deployment, thereby enabling longevity without retraining.

Inference-Time Caching and Feature Reuse. Balasubramanian et al. [Balasubramanian et al. 2021] introduced *GATI*, which learns to predict layer outputs from cached representations, achieving up to 7.7× latency reduction in cloud settings. However, as it relies on auxiliary models and continuous retraining, it is unsuitable for low-power IoT nodes. In contrast, our cache stores numerically exact affine mappings from previous activations, requiring no additional learning. Feature reuse has also been explored in vision. DeepCache [Xu et al. 2018] caches convolutional feature maps across similar video frames, achieving ∼18% speedup and 20% energy savings on mobile devices. These approaches exploit spatial or temporal locality but depend on heuristic frame matching. Our region-level cache instead operates on the network's intrinsic linear structure and guarantees exact reuse for repeated activation patterns.

Conditional Computation. Early-exit networks and MoE models achieve conditional computation by activating only part of the network—either exiting early or selecting a subset of experts per input—trading slight accuracy loss for speed [Addad et al. 2023; Bolukbasi et al. 2017; Shazeer et al. 2017; Teerapittayanon et al. 2016]. While effective, both require architectural changes or retraining. In contrast, our approach performs *exact computation reuse* within an unmodified pre-trained model and can complement such dynamic-inference methods.

Lookup-Table Accelerators. Lookup-table (LUT) methods replace arithmetic with indexed lookups. *MADDNESS* [Blalock and Guttag 2021] precomputes subspace dot products to approximate matrix multiplications, reducing compute at the cost of minor accuracy loss. *LUT-NA* [Sen et al. 2024] performs exact digital inference via low-precision LUT operations, achieving up to 3.3× lower energy and 29× smaller area without retraining. These hardware optimizations complement our method: LUTs accelerate every operation, while caching avoids redundant ones. Both align with inference-time, model-agnostic efficiency gains.

Overall, prior work accelerates inference through data locality, architectural sparsity, or hardware substitution. Our method specifically applies linear-region caching to time-series neural inference on IoT devices—reusing exact affine mappings across repeated activation patterns to reduce compute and energy consumption under hardware aging, thereby extending device longevity without retraining or model modification.

3 Background

This section introduces the notation and theoretical concepts underlying ReLU-based feedforward neural networks used in this work.

Network structure. We consider a neural network $f_\theta : \Omega \to \mathbb{R}^C$ with trainable parameters θ. The domain $\Omega \subset \mathbb{R}^D$ denotes the input space of dimension D, and the output $\mathbb{R}^C$ corresponds to the unnormalized class logits. Each hidden layer $l \in \{1, \dots, L\}$ consists of d_l neurons, parameterized by a weight matrix $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ and bias vector $\mathbf{b}^{(l)} \in \mathbb{R}^{d_l}$. Given an input $\mathbf{x} \in \Omega$, the forward pass proceeds recursively as

$$\mathbf{p}^{(l)} = W^{(l)}\mathbf{q}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{q}^{(l)} = \sigma\left(\mathbf{p}^{(l)}\right), \tag{1}$$

where $\mathbf{q}^{(0)} := \mathbf{x}$ denotes the input layer. The final layer is linear:

$$f_\theta(\mathbf{x}) = W^{(L+1)}\mathbf{q}^{(L)} + \mathbf{b}^{(L+1)}. \tag{2}$$

The resulting vector $f_\theta(\mathbf{x})$ contains one logit per class, typically transformed by a softmax function for probabilistic classification.

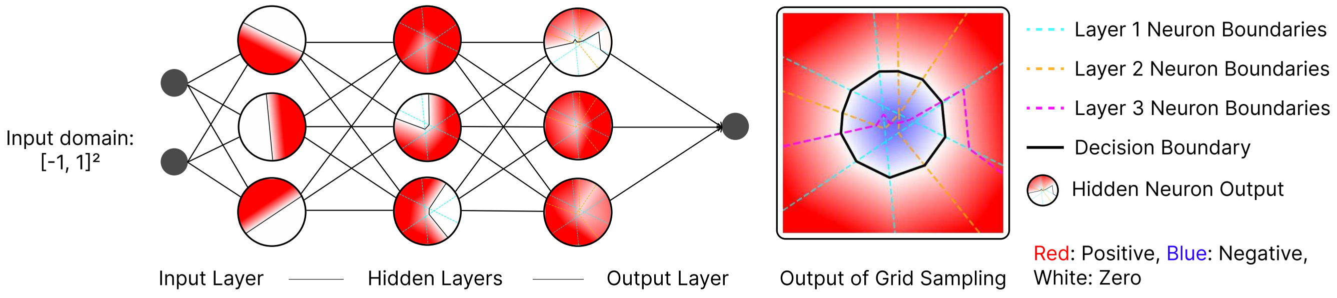

Input domain:
[-1, 1]²
Input Layer
Hidden Layers
Output Layer
Output of Grid Sampling
Layer 1 Neuron Boundaries
Layer 2 Neuron Boundaries
Layer 3 Neuron Boundaries
Decision Boundary
Hidden Neuron Output
Red: Positive, Blue: Negative, White: Zero

Figure 1: Linear regions in a ReLU MLP trained on a synthetic signed distance dataset of a circle. Dashed lines denote distinct affine regions of the ReLU network. Red indicates positive, blue negative values and white zero.

Piecewise linearity. The ReLU activation [Nair and Hinton 2010] is defined as

$$\sigma(x) = \max(0, x), \tag{3}$$

and is linear on each side of its non-differentiable point at zero. Because every layer in Eqs. (1)–(2) is an affine map followed by a ReLU, their composition is also affine on local subsets of the input space. Thus, any feedforward ReLU network (in this case f_θ) represents a continuous piecewise-linear function [Arora et al. 2018; Hanin and Rolnick 2019]. Each neuron defines a hyperplane

$$\{\mathbf{x} \in \mathbb{R}^D \mid \mathbf{p}_i^{(l)}(\mathbf{x}) = 0\}, \tag{4}$$

that partitions the space into two half-spaces where the neuron is *active* ($\mathbf{p} > 0$) or *inactive* ($\mathbf{p} \leq 0$). The pattern of active neurons determines a *region of linearity*, within which the network behaves as a single affine mapping [Poole et al. 2016; Raghu et al. 2017].

Partitioning into linear regions. The intersection of all neuron hyperplanes across layers subdivides Ω into convex polytopes or *linear regions* [Hanin and Rolnick 2019; Montúfar et al. 2014]. Each region corresponds to a unique binary activation pattern across neurons, and the network is affine inside that region:

$$\widetilde{f}(\mathbf{x}) = \widetilde{W}\,\mathbf{x} + \widetilde{\mathbf{b}}. \tag{5}$$

Here, $\widetilde{W}$ and $\widetilde{\mathbf{b}}$ are obtained by collapsing all active submatrices of $W^{(l)}$ and corresponding biases through the network hierarchy. The affine form in Eq. (5) remains valid within the region C and changes only when the input crosses a neuron boundary, activating or deactivating a unit. Figure 1 shows the regions for a two-dimensional example trained on a synthetic dataset representing a signed distance function of a circle. The input domain is sampled on a grid; each circle represents the output of the neuron after applying ReLU. Dashed lines indicate the neuron boundaries (zero-level set) that separate the piecewise linear parts.

Geometrically, each hidden neuron introduces a half-space constraint, and deeper layers iteratively refine the subdivision of Ω. The number of regions grows combinatorially with depth: a deeper ReLU network can carve exponentially more regions than a shallow one with a similar parameter count [Poole et al. 2016; Raghu et al. 2017]. Montúfar *et al.* [Montúfar et al. 2014] established lower bounds on this growth, showing that an L-layer network with n neurons per layer can realize on the order of $\Omega((n/n_0)^{(L-1)n_0} n^{n_0})$ distinct regions, where n_0 is input dimension. Telgarsky [Telgarsky 2016] further demonstrated that depth yields exponentially more expressive power: there exist functions representable by a depth-$\Theta(k^3)$ network that cannot be approximated by any network of depth $O(k)$ unless it has exponentially many neurons. These findings formalize the intuition that deeper networks can form more complex decision boundaries through hierarchical region composition. For classification, transitions between classes occur at the boundaries where two logits are equal, forming a piecewise-linear approximation of the underlying decision manifold [Hanin and Rolnick 2019].

4 Hierarchical Hypersphere Caching

The piecewise-linear structure of ReLU networks partitions the input space into a collection of linear regions where the network behaves as an affine function [Arora et al. 2018; Hanin and Rolnick 2019; Montúfar et al. 2014]. This enables reusing previously computed linearizations for any new input that falls into a region encountered before [Joyce and Verschelde 2025]. Each time a ReLU MLP is evaluated at a point $\mathbf{x}_0$, the pattern of active and inactive neurons defines a local *linear cell* (a convex polytope) within which the network acts as a single affine mapping $\widetilde{f}(\mathbf{x}) = \widetilde{W}\mathbf{x} + \widetilde{\mathbf{b}}$. Although a deep network can, in theory, partition its domain into an exponential number of such regions [Montúfar et al. 2014], empirical studies show that real ReLU networks realize far fewer distinct activation patterns [Hanin and Rolnick 2019]. This redundancy suggests that large portions of the input space share the same affine transformation, creating opportunities for computational reuse. Joyce and Verschelde [Joyce and Verschelde 2025] formalized this insight by showing that each unique ReLU activation pattern corresponds to a distinct affine mapping that can be memoized for faster inference.

Hypersphere caching. Our proposed *Hypersphere Cache* associates each cached linearization with the largest *inscribed hypersphere* centered at $\mathbf{x}_0$ that remains fully inside the corresponding linear region. Its radius is determined by the minimum Euclidean distance to any neuron activation boundary. Geometrically, this sphere defines the maximal region around $\mathbf{x}_0$ in which all ReLU activations remain constant. Hence, if a future query point $\mathbf{x}$ lies within an existing sphere, the stored affine mapping is reused directly (a *cache hit*); otherwise, a new linearization is computed and stored (a *cache miss*), analogous to region-based caching [Joyce and Verschelde 2025] and feature reuse approaches such as GATI [Balasubramanian et al. 2021] or LUT-based inference methods [Blalock and Guttag 2021; Sen et al. 2024].

Cache representation. The cache maintains tuples of the form $(\mathbf{x}_0, r, W_{\text{final}}, \mathbf{b}_{\text{final}})$, where $\mathbf{x}_0 \in \mathbb{R}^D$ is the hypersphere center, $r \in \mathbb{R}^+$ its radius, and $W_{\text{final}} \in \mathbb{R}^{C\times D}$, $\mathbf{b}_{\text{final}} \in \mathbb{R}^C$ define the affine mapping. Each entry satisfies $f_\theta(\mathbf{x}) = W_{\text{final}}\mathbf{x}+\mathbf{b}_{\text{final}}$ for all $\mathbf{x}$ within the hypersphere. When a new query arrives, the cache searches for a containing sphere and either reuses the stored linearization or computes a new one.

Hierarchical lookup strategy. To achieve fast lookup times, we use a hierarchical three-stage search exploiting temporal and spatial locality:

(1) **Last-hit check (O(1))**: The previously used entry `last_idx` is checked first, as consecutive inputs often lie in the same region. If $\|\mathbf{x} - \mathbf{x}_0\|_2 < r$, the cached linearization is reused immediately.
(2) **Recent insertions (O(k))**: Newly added spheres not yet in the spatial index are searched next, enabling immediate reusability.
(3) **Spatial index (O($\log n$))**: Older cached entries are organized in a ball tree using Euclidean distance. The depth of this tree grows logarithmically. Given the maximum stored radius $r_{\max}$, the index retrieves all hyperspheres with centers within $r_{\max}$ of $\mathbf{x}$. Only a small subset of candidates are then distance-checked for containment.

This hierarchical structure combines constant-time reuse in common cases with logarithmic-time spatial search in the general case, yielding fast average query performance even as the cache grows.

Cache updates and rebuilds. On a cache miss, the network performs a full forward pass to compute a new linearization at $\mathbf{x}$. The valid radius is determined as $r = \min_{l,i} |r_{l,i}|$, ensuring conservative containment within the linear region. The new entry is appended to the cache and integrated into the ball tree once the number of pending insertions exceeds a threshold, amortizing the $O(n \log n)$ cost of index maintenance.

Illustration. Fig. 2 visualizes a two-dimensional example. Cyan, orange, and magenta lines denote neuron boundaries from three hidden layers, subdividing the input into convex linear cells. Dotted circles indicate cached hyperspheres. Orange circles mark new linearizations (cache misses), while blue squares indicate reuse events (cache hits). Larger hyperspheres appear in stable regions far from activation boundaries, while smaller spheres cluster near decision boundaries where neuron states change frequently. This behavior mirrors results from activation pattern analyses [Hanin and Rolnick 2019; Raghu et al. 2017] and illustrates how the cache adapts to local model smoothness.

5 Evaluation & Results

5.1 Datasets

We evaluate our method on four publicly available human activity recognition (HAR) datasets: *WISDM*, *MHEALTH*, *PAMAP2*, and *UCI-HAR*. A summary of their key characteristics is shown in Table 1.

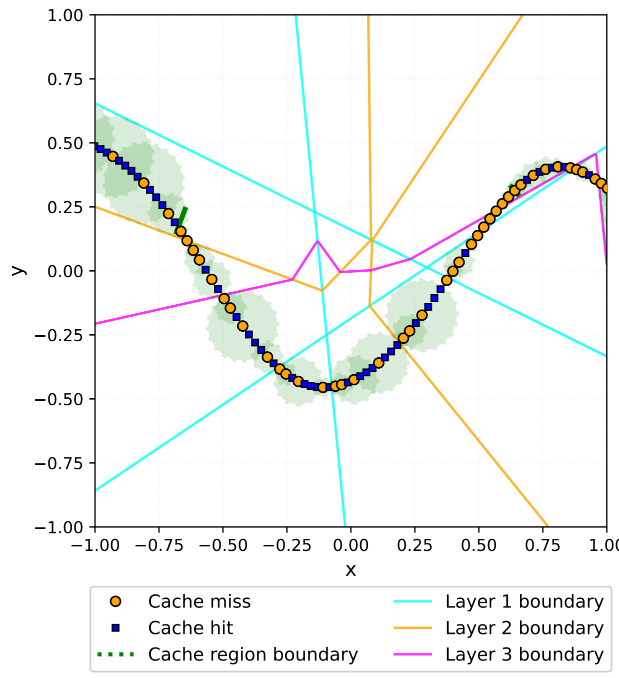


Figure 2: Caching visualization of a model with 2 input neurons.

WISDM. The WISDM dataset [Kwapisz et al. 2011] contains smartphone and smartwatch motion-sensor recordings (accelerometer and gyroscope) from 51 subjects performing everyday activities. We use the raw smartphone accelerometer signals sampled at 20 Hz and focus on non-hand-oriented activities (walking, jogging, climbing stairs, and standing). Sensor data are segmented using 2.5 s sliding windows (50 readings per window) with 50 % overlap. For each axis, five statistical features (mean, standard deviation, minimum, maximum, median) are extracted to form 15-dimensional feature vectors.

MHEALTH. The MHEALTH dataset [Baños et al. 2014] includes body motion and vital sign recordings from ten volunteers equipped with three sensors (chest, right wrist, left ankle), each providing 23 readings at 50 Hz. We consider six activities (standing, sitting, walking, climbing stairs, jogging, and running). Windows of 2 s (100 readings) with 50 % overlap are used, and features are extracted analogously to WISDM.

Table 1: Summary of datasets used for evaluation.

Dataset	Input Features	Activities	Samples
WISDM	15	5	53,475
MHEALTH	115	7	4,290
PAMAP2	124	6	24,600
UCI HAR	561	6	10,299

PAMAP2. The Physical Activity Monitoring (PAMAP2) dataset contains recordings from nine subjects wearing three inertial measurement units (IMUs) and a heart-rate monitor [Reiss and Stricker 2012]. The IMUs sample at 100 Hz and the heart-rate sensor at 9 Hz. We focus on lying, sitting, standing, walking, and ascending/descending stairs. Feature extraction follows the same procedure using 1 s windows (100 readings) with 50 % overlap.

UCI-HAR. The UCI-HAR dataset [Anguita et al. 2013] comprises accelerometer and gyroscope signals from 30 volunteers performing six activities (walking, walking upstairs/downstairs, sitting, standing, and lying). The smartphone-mounted sensors sample at 50 Hz. Preprocessed windows of 2.56 s (128 readings) with 50 % overlap are provided, each represented by 561 pre-extracted features.

Train–test splits. For UCI-HAR, the predefined subject split is used. The remaining datasets are partitioned by subject: approximately 20 % of subjects are held out for testing, while the others form the training set. Features are extracted per subject, and test data preserve temporal continuity within each subject—crucial for evaluating our caching algorithm under realistic, continuous trajectories.

5.2 Models

We employ MLP architectures with ReLU activations for IoT time-series classification. The choice of ReLU is intentional: as a piecewise-linear activation, it partitions the input space into exponentially many linear regions [Raghu et al. 2017], which directly enables our region-based caching optimization.

All models use ReLU activations in the hidden layers and a linear output layer that produces logits for softmax-based multi-class activity classification. The input dimensionality corresponds to the feature size of each dataset: WISDM (15), MHEALTH (115), PAMAP2 (124), and UCI-HAR (561).

Training Configuration. Models are trained using the Adam optimizer [Kingma and Ba 2017], for 100 epochs, a learning rate of 0.001, batch size 64, and cross-entropy loss. The model with the lowest validation loss is saved.

Hyperparameter Optimization. A grid search varying network width (8–512 hidden units) and depth (1–5 layers) identifies optimal architectures for each dataset. Figure 3 shows validation accuracy as a function of hidden-layer width, averaged across depths.
The best configurations are: WISDM (256 units), MHEALTH (64 units), PAMAP2 (256 units), and UCI_HAR (128 units). Datasets requiring fewer layers compensate with wider hidden dimensions.

Figure 4 presents validation accuracy versus network depth, averaged over all widths.
The selected architectures are: WISDM (3 layers), MHEALTH (4 layers), PAMAP2 (1 layer), and UCI_HAR (1 layer). Simpler datasets achieve peak accuracy with shallow networks, whereas WISDM and MHEALTH benefit from deeper architectures capturing more complex feature interactions.

5.3 Caching

We evaluate the proposed hypersphere caching mechanism across four HAR datasets. The cache exploits the local linearity of ReLU networks by storing affine approximations valid within hyperspherical regions around previously seen inputs, as defined in Section 4. Evaluation is performed in an online inference setting over continuous test sequences, where each incoming sample either reuses a cached linearization or triggers a new one.

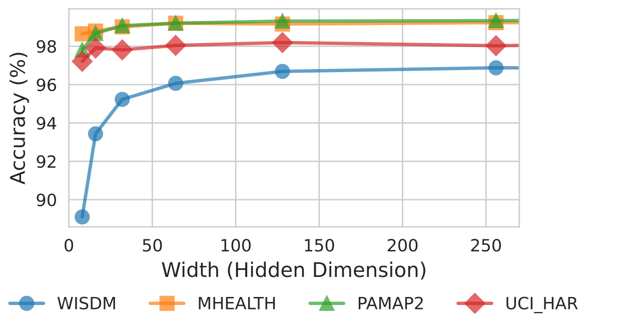


Figure 3: Accuracy vs. hidden dimension width, averaged across depths.

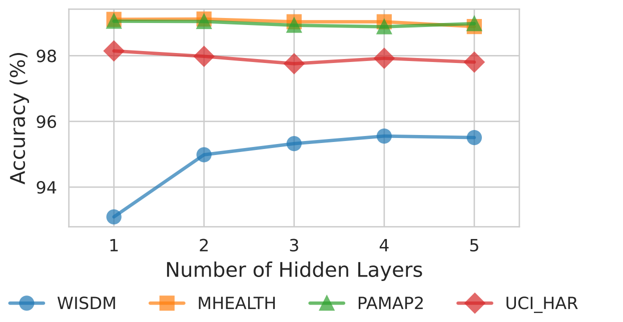


Figure 4: Accuracy vs. number of hidden layers, averaged across widths.

Cache construction and quantile-based radius selection. For each cached point, we compute the distance to the nearest neuron decision boundary at each layer, as defined in Section 4, i.e., the minimum Euclidean distance to any activation boundary derived from the layer's linearization at that point. These distances define the maximum radius within which the linearization remains exact. We store these radii and sort them to create a distribution, from which we select cache radii based on quantiles (10th to 40th percentile). This quantile-based selection allows us to trade off between cache hit rate and approximation quality.

Cache hit rates across datasets. Figure 5 shows the cache hit rate as a function of the selected radius quantile. WISDM exhibits the highest cache utilization, with hit rates increasing from 34% at the 10th percentile to 55% at the 40th percentile.
This indicates that over half of incoming samples fall within previously cached regions when using larger radii. MHEALTH and

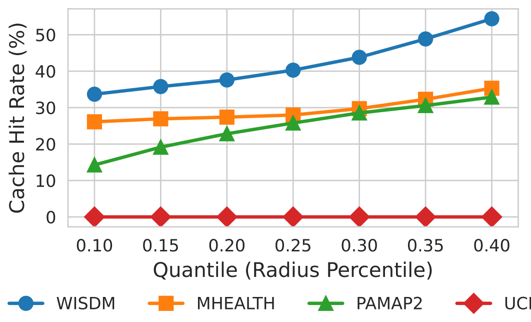


Figure 5: Cache hit rate vs. radius quantile. Higher quantiles correspond to larger hypersphere radii, increasing the likelihood of cache reuse.

PAMAP2 show moderate cache effectiveness, with hit rates climbing from 27% and 15% respectively at the smallest radii to approximately 35% and 33% at the largest. Notably, UCI-HAR shows negligible cache reuse across all quantiles, suggesting its input patterns do not revisit similar regions during inference. The cache hit rate correlates strongly with the input dimensions. Table 1 shows that WISDM uses 15 input dimensions, whereas UCI HAR has 561 input dimensions.

Preservation of model accuracy. Expanding the cache radius to enable limited extrapolative reuse does not degrade classification accuracy. Figure 6 shows that accuracy remains effectively constant across all quantiles:

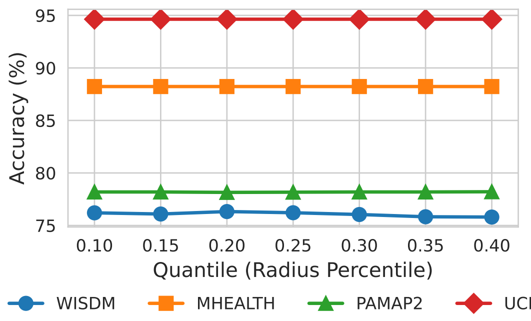


Figure 6: Classification accuracy vs. radius quantile. Accuracy remains stable across all quantile selections, indicating no degradation from extrapolative reuse.

UCI-HAR 95 %, MHEALTH 88 %, PAMAP2 78 %, and WISDM 76 %, all within ±0.1 percentage points. This stability confirms that cached affine approximations remain valid even slightly beyond their theoretical region boundaries.

Computational savings. Figure 7 quantifies the FLOP reduction achieved through caching, comparing the computational cost of using cached linear models versus full forward passes.
Reduction patterns mirror hit rates: WISDM achieves 34-55% fewer FLOPs, while MHEALTH and PAMAP2 show moderate savings

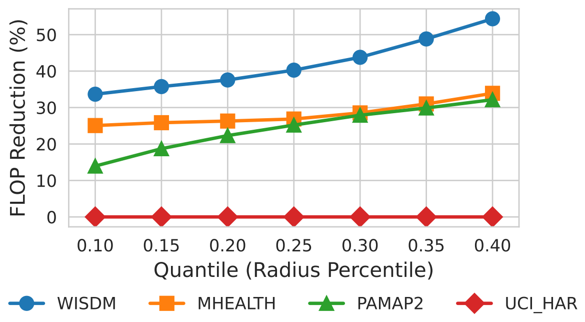


Figure 7: FLOP reduction vs. radius quantile. Values represent computational savings from using cached linear approximations instead of full forward passes.

around 25-35% at higher quantiles. These measurements represent the theoretical speedup from replacing full forward passes with single matrix-vector multiplications, excluding cache lookup overhead and linearization costs. Optimizing these auxiliary operations remains future work, but the results demonstrate substantial potential for computational savings in embedded deployments.

Implications for embedded deployment. These results confirm that ReLU networks processing temporal sensor data repeatedly activate a limited set of linear regions, enabling efficient caching. The hypersphere cache yields up to 55 % theoretical computational reduction with no accuracy loss, underscoring its suitability for resource-constrained IoT devices requiring long-term, energy-efficient inference.

6 Threats to Validity

While our experiments demonstrate the potential energy savings of using caching to reduce FLOPs on microcontrollers without hardware FPUs, there are several threats to the validity of our results:

- **Limited practical validation:** Our FLOP reduction estimations only give a rough estimate for compute and energy savings. More practical experiments are needed, where the neural network models and caching mechanism are deployed on real hardware, and energy consumption is measured for users performing activities similar to those in the datasets. Such measurements would provide more accurate and reliable validation of the energy savings.
- **Feature dimensionality and model scaling:** Our current approach relies on a relatively small number of input features derived from sensor windows. Scaling to higher-dimensional input spaces or more complex features may increase the number of FLOPs, potentially offsetting the energy savings from caching. Additionally, large input dimensions could require larger neural networks or more hidden layers to maintain accuracy, which may further impact energy consumption. The cache size is also dependent on the dimensions; the additional RAM needed for high-dimensional inputs could force one to flush the cache

more often, leading to a higher number of cache misses and further amplifying the problem.

- **Cache effectiveness:** The benefits of caching depend on temporal redundancy in the sensor data and the chosen radius quantile, both impacting the cache hit rate. Different activity patterns or datasets may reduce redundancy, diminishing the energy savings.

Addressing these threats in future work will strengthen the validity of our conclusions and provide a more accurate assessment of the trade-offs involved in reducing floating-point operations for embedded neural network inference.

7 Conclusion

This work introduced *Neural Caching*, a region-based inference mechanism that leverages the piecewise-linear structure of ReLU networks to accelerate neural inference on resource-constrained IoT devices. By caching affine mappings corresponding to previously activated ReLU regions, the proposed *Hypersphere Cache* enables subsequent queries to be resolved through a single matrix multiplication instead of a full forward pass. We demonstrated that this approach yields a consistent reduction in necessary floating-point operations, given low input dimensionality, while maintaining classification fidelity across multiple HAR datasets.

Unlike compression or quantization methods, our technique operates entirely at inference time, requiring no retraining or architectural modifications. The empirical results confirm that ReLU-MLPs revisit a limited subset of activation patterns during continuous sensing, enabling efficient reuse without accuracy degradation. Moreover, because caching reduces the number of active operations and memory accesses, it directly contributes to lower energy consumption and thus extends hardware longevity—an essential goal for long-lived IoT deployments.

Future work includes integrating the cache into mixed-precision and hardware-accelerated inference pipelines, exploring adaptive cache replacement strategies under dynamic workloads, and extending the approach to convolutional and transformer architectures. By coupling neural caching with emerging low-power accelerators, sustainable and high-performance AI inference on embedded systems becomes increasingly feasible, advancing the vision of autonomous and energy-aware IoT intelligence.

References

Youva Addad, Alexis Lechervy, and Frédéric Jurie. 2023. Multi-Exit Resource-Efficient Neural Architecture for Image Classification with Optimized Fusion Block. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*. IEEE, Paris, France, 1486–1491. https://openaccess.thecvf.com/content/ICCV2023W/

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. 2013. A public domain dataset for human activity recognition using smartphones.. In *Esann*, Vol. 3. 3–4.

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. 2018. Understanding Deep Neural Networks with Rectified Linear Units. arXiv:1611.01491 [cs.LG] https://arxiv.org/abs/1611.01491

Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. 1997. Locally weighted learning. *Artificial intelligence review* 11, 1 (1997), 11–73.

Arjun Balasubramanian, Adarsh Kumar, Yuhan Liu, Han Cao, Shivaram Venkataraman, and Aditya Akella. 2021. Accelerating Deep Learning Inference via Learned Caches. *ArXiv* abs/2101.07344 (2021). https://api.semanticscholar.org/CorpusID:231639254

Oresti Baños, Rafael García, Juan Antonio Holgado Terriza, Miguel Damas, Héctor Pomares, Ignacio Rojas, Alejandro Saez, and Claudia Villalonga. 2014. mHealthDroid: A Novel Framework for Agile Development of Mobile Health Applications. In *International Workshop on Ambient Assisted Living and Home Care*. https://api.semanticscholar.org/CorpusID:9757468

Davis Blalock and John Guttag. 2021. Multiplying matrices without multiplying. In *International Conference on Machine Learning*. PMLR, 992–1004.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. In *International conference on machine learning*. PMLR, 527–536.

Boris Hanin and David Rolnick. 2019. Deep ReLU networks have surprisingly few activation patterns. *Advances in Neural Information Processing Systems* 32 (2019).

Johnny Joyce and Jan Verschelde. 2024. Algebraic representations for faster predictions in convolutional neural networks. In *International Workshop on Computer Algebra in Scientific Computing*. Springer, 161–177.

Johnny Joyce and Jan Verschelde. 2025. Computing Linear Regions in Neural Networks with Skip Connections. *arXiv preprint arXiv:2509.15441* (September 2025). https://arxiv.org/abs/2509.15441

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] https://arxiv.org/abs/1412.6980

Daniël Kraak, Mottaqiallah Taouil, Said Hamdioui, Pieter Weckx, Francky Catthoor, Abhijit Chatterjee, Adit Singh, Hans-Joachim Wunderlich, and Naghmeh Karimi. 2018. Device aging: A reliability and security concern. In *2018 IEEE 23rd European Test Symposium (ETS)*. IEEE, 1–10.

Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. 2011. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.* 12, 2 (March 2011), 74–82. doi:10.1145/1964897.1964918

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. 2014. On the number of linear regions of deep neural networks. *Advances in neural information processing systems* 27 (2014).

Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 807–814.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. 2016. Exponential expressivity in deep neural networks through transient chaos. In *NeurIPS*.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. 2017. On the Expressive Power of Deep Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 2847–2854. https://proceedings.mlr.press/v70/raghu17a.html

Attila Reiss and Didier Stricker. 2012. Introducing a New Benchmarked Dataset for Activity Monitoring. *2012 16th International Symposium on Wearable Computers* (2012), 108–109. https://api.semanticscholar.org/CorpusID:10337279

Ovishake Sen, Chukwufumnanya Ogbogu, Peyman Dehghanzadeh, Janardhan Rao Doppa, Swarup Bhunia, Partha Pratim Pande, and Baibhab Chatterjee. 2024. Look-Up Table based Neural Network Hardware. arXiv:2406.05282 [cs.AR] https://arxiv.org/abs/2406.05282

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2464–2469.

Matus Telgarsky. 2016. Benefits of depth in neural networks. In *Conference on learning theory*. PMLR, 1517–1539.

Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. 2018. Deepcache: Principled cache for mobile deep vision. In *Proceedings of the 24th annual international conference on mobile computing and networking*. 129–144.

Syed Faizan Alam Zaidi, Junaid Arshad, and Syed Farrukh Alam Zaidi. 2024. A Review on Performance Prediction Models for Battery Life in IoT Networks. *International Journal of Multidisciplinary Sciences and Engineering (IJMSE)* 15, 3 (July 2024), 13–19. https://www.ijmse.org/Volume15/Issue3/paper4_15_3.pdf

Hybrid IoT Platform Architectures for Flexible and Longstanding Deployments

Katarina Vuknić
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
katarina.vuknic@fer.unizg.hr

Mario Kušek
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
mario.kusek@fer.unizg.hr

Ivana Podnar Žarko
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
ivana.podnar@fer.unizg.hr

Abstract

The paper analyzes hybrid IoT platform architectures that integrate cloud-based and edge-based solutions. While cloud platforms offer scalability and a centralized global view, edge platforms provide low latency, enhanced privacy, and local autonomy. By combining them, hybrid systems can leverage both local performance aspects (e.g., real-time actuation and device management) and global centralized processing at the cloud level. The edge is enabled to share filtered and aggregated local views with the cloud, while local changes of IoT devices at the edge level do not affect the cloud level. Such architectures are thus adequate for longstanding IoT deployments. We review existing strategies for integrating IoT devices with cloud platforms and introduce a flexible, edge-centric integration approach which simplifies device replacement procedures and offers configurable data synchronization with the cloud. We validate the approach through a case study on a hybrid smart home solution that integrates an open-source edge platform with a commercial cloud-based platform to manage and process home automation data across large-scale deployments.

Keywords

interoperability, intermediary edge gateway, IoT platform federation

1 Introduction

The integration of local and cloud platforms aims to combine the strengths of edge and cloud computing to achieve a more resilient, efficient and flexible system. Cloud platforms typically support application-layer communication protocols, such as MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol), or HTTP (HyperText Transfer Protocol). However, they often lack native support for device-level communication standards commonly used in home automation, industrial environments, or field monitoring, such as Zigbee, LoRa (Long Range), or BLE (Bluetooth Low Energy), which operate primarily on the link and physical layers of the communication stack. Consequently, most cloud-based IoT platforms do not provide direct drivers or plugins for such non-IP devices. In contrast, edge-based platforms, often open-source (e.g., Home Assistant [6], openHAB [22]), provide integration plugins for various low-level or lightweight protocols and device standards.

Within hybrid local–cloud architectures, these edge platforms or intermediate gateways can perform protocol translation, converting heterogeneous device messages into standardized data formats such as MQTT payloads or RESTful JSON objects. Such standardized representations can then be transmitted to the cloud for further processing including analytics, visualization, machine learning (ML), or digital twin synchronization. Collecting and pre-processing data locally before forwarding it to the cloud significantly reduces latency compared to cloud-only architectures, while also enhancing privacy by keeping raw data closer to the user. This design approach allows end users to benefit from the real-time-like responsiveness on the edge and operational continuity provided by the edge-based platform, as well as from the centralized management and advanced data analytics capabilities of the cloud platform. In cases where the edge and cloud components are developed by different vendors, an edge gateway layer should be introduced. This gateway acts as a protocol and data format translator, ensuring syntactic, semantic, and organizational interoperability. Firstly, with such setup, edge-based platform ensures continuous service to the user even when the connection to the cloud is temporarily unavailable. Secondly, data collected at the edge-based platform can be filtered before being sent to the cloud, reducing the processing load on the cloud and enhancing user privacy, as some data remains only locally. Thirdly, certain rules and automation can be executed locally for quick response, while more complex or coordinated operations can be managed from the cloud. For example, a washing machine connected to the edge-based platform may have a local rule to operate when electricity is low-cost, while the cloud can aggregate data from multiple households and determine an optimal distribution of appliance usage across a neighborhood, ensuring that not all machines start simultaneously at peak low-cost hours. Finally, the architecture should guarantee longevity and flexibility, enabling the replacement or migration of cloud providers without disrupting the overall system.

2 Related work

The manner in which an IoT platform obtains the state of end devices can be implemented at multiple levels, depending on the protocols and system architecture. In some cases, the device itself sends a notification of a change as soon as it occurs (push model), which allows for a fast response and lower latency. Alternatively, the platform can periodically poll the device to retrieve the current state (polling model), which is simpler to implement but introduces latency and additional network load [28]. The second dimension refers to where the communication takes place: within the local network, where the platform can manage devices even in the event of a connection failure, or via the manufacturer's cloud, where the platform depends on the availability of an external service. Combining these characteristics, we obtain four typical patterns of device-platform integration: local data push, local state polling, cloud notifications, and cloud data retrieval. This categorization [19] [21] is valid regardless of the specific platform and can also be applied to open-source solutions such as Home Assistant and openHAB [29].

Recent advances in IoT platform architectures have emphasized the importance of hybrid edge-cloud computing approaches that address the limitations of purely cloud-based solutions [2, 3]. The integration of edge computing with IoT devices has facilitated the creation of distributed computing architectures capable of handling massive data volumes generated by interconnected devices [30]. This convergence enables efficient data aggregation, analysis, and decision-making at the network's edge, reducing the burden on centralized cloud infrastructure and optimizing resource utilization [14].

2.1 Semantic Interoperability and Platform Integration

Generic platforms typically lack formal information models, allowing flexibility in connecting devices and defining data formats and protocols. This is especially convenient for dealing with a wide range of vendor-specific devices and enabling unified control and automation within a heterogeneous smart environment. The challenge of semantic interoperability has become increasingly important as IoT ecosystems expand beyond individual vendor silos. Recent research has focused on developing semantic interoperability support systems (SISS) that enable the generation of gateways or translator components between disparate data models [12, 13].

The FIWARE ecosystem represents a significant advancement in standardizing IoT platform interoperability through its NGSI-LD information model and API [9, 10]. FIWARE provides open-source components, called generic enablers, for building IoT platforms based on formal and de-facto standards. The NGSI-LD standard, formalized by the ETSI Industry Specification Group on Context Information Management, forms the basis of interoperability between core FIWARE components [9, 33].

Beyond the integration of individual end devices, we consider the integration of one IoT platform with another, where a local IoT platform (e.g., Home Assistant or openHAB) serves as a bridge to the cloud. An example of this approach is the connection between Home Assistant and the IBM Watson IoT Platform [8], for which Home Assistant provides an official integration. Through this integration, the platform registers directly as a gateway on Watson IoT and exchanges telemetry data and control commands via the MQTT protocol, without requiring any additional middleware. Also, there are similar integrations with Google Cloud [7] and Amazon Web Services [4], or modular cloud components such as Azure [5].

Recent studies have demonstrated the practical implementation of such integrations in real-world deployments. For instance, research on smart home automation systems using Home Assistant has shown significant improvements in energy efficiency, with systems achieving up to 22.5% reduction in overall energy consumption [11]. These implementations utilize various communication protocols including Zigbee, Z-Wave, and Wi-Fi to ensure seamless device interoperability [26, 29].

2.2 Cross-Platform Federation and Middleware Solutions

On the other hand, most cloud platforms are not natively supported by local IoT platforms, which necessitates intermediary solutions. A solution like SymbIoTe [24, 25], a platform-agnostic interoperability framework, works at a higher, cross-platform level, where different IoT platforms are connected via a common semantic model and standardized APIs. The SymbIoTe approach offers mediation services for search and controlled access to IoT resources across platforms in a uniform way, providing an IoT Portal with registration and search capabilities using semantic web technologies [25].

This makes it possible for the resources of one platform to be available to another, without the latter having to know internal protocols or implementations. In contrast, an approach utilizing edge gateway, e.g. Node-RED acting as a gateway at the local side [1, 17], operates at the local level, where it translates device protocols (SISS) and enables them to connect to the cloud or a single target platform. This is often simpler and more practical in real implementations, as it enables fast device integration and reliable management of local resources without requiring complex semantic models.

Recent research has emphasized the importance of publish/subscribe broker clustering for large-scale IoT applications, proposing hierarchical edge-cloud models that use efficient two-tier routing schemes to alleviate latency and scalability issues [23]. These approaches leverage proximate edge brokers strategically deployed in edge networks for data delivery services, demonstrating the effectiveness of distributed architectures in handling geo-distributed IoT devices.

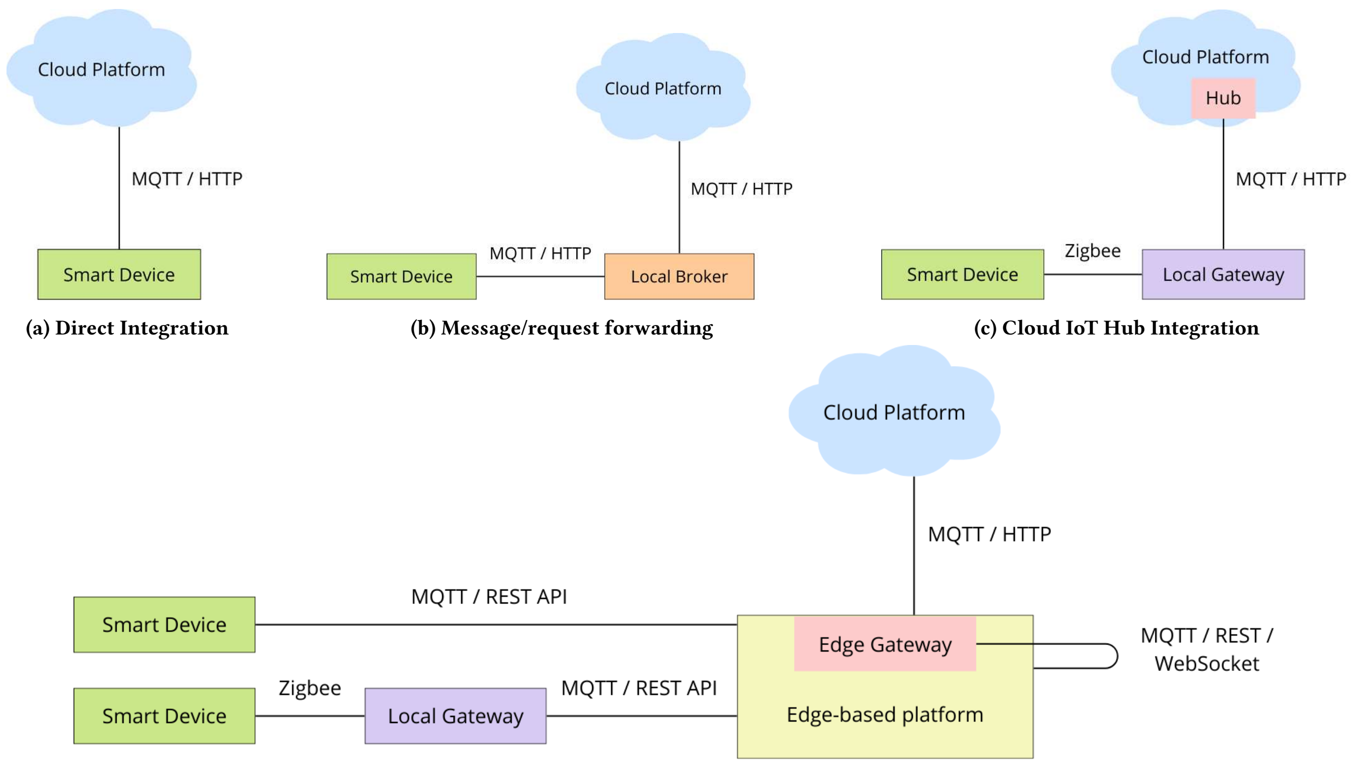
The integration challenges extend beyond technical interoperability to include organizational and security considerations. Recent studies have highlighted the importance of secure semantic interoperability, particularly concerning the privacy implications of linking data across different IoT platforms [18]. These security considerations become increasingly critical as IoT deployments scale and involve multiple stakeholders across different domains.

3 Connecting devices to the cloud-based platform

There are four ways in which end devices can be connected to the IoT platform hosted in the cloud, either directly or via an edge-based platform acting as middleware, and with the varying components in between.

3.1 Direct (custom) integration

Directly integrating the end device to the platform in the cloud doesn't require any intermediary, as shown in Figure 1 a). However, if the cloud platform doesn't offer a ready-made protocol, content format and information model compatible with the device, it has to be adapted manually (custom API, driver, SDK). Security measures,



(a) Direct Integration **(b) Message/request forwarding** **(c) Cloud IoT Hub Integration**

(d) Intermediary Edge Gateway at the Edge-based Platform

Figure 1: Connecting devices to the cloud-based platform

including TLS certificates, authentication and disconnection handling, must be implemented on the device. In case of no Internet service, the device is left with no local control. This approach is convenient when the device has enough resources to perform the entire integration. However, this approach lacks flexibility for local actuation, as it fully relies on the cloud for rule-based instructions, thereby introducing inherent latency. A practical example of this model can be found in smart assistants such as Amazon Alexa, which connects directly to the AWS platform in the cloud.

3.2 Message/request forwarding

The idea is to have some component in local/edge environment that receives data from IoT devices in local environment and forwards them to cloud platform. One solution is MQTT-forwarding where the local broker acts as an intermediary that forwards messages from end device to a connected cloud MQTT broker, as shown in Figure 1 b). This model is lightweight and relatively simple to deploy, but it has inherent limitations. Message transformation or protocol adaptation is not provided, so devices must produce payloads in a format already compatible with the cloud subscriber. Furthermore, system operation is fully dependent on continuous network connectivity, and this approach is generally restricted to MQTT-capable devices, excluding HTTP or other protocol-based devices. Common implementations of local broker include Mosquitto, which provides reliable message queuing and forwarding for such architectures. Another solution is to use Eclipse Kura in local edge environment [15]. Eclipse Kura is used for protocol translation, and secure connectivity between field devices and cloud platforms. The drawback is that it is service that could be run as part of edge gateway and it is not ready server to install.

3.3 Cloud IoT Hub integration

Compared to direct device-to-cloud integration, where the device itself must implement the protocol expected by the cloud, the IoT hub provides format and protocol translation in the cloud, thus simplifying interoperability. The hub in the cloud acts as a central entry point for all connected devices, as shown in Figure 1 c). Here, the local gateway serves as a protocol translator for devices that use communication protocols without native Internet support. This approach does not support offline operation, since all communication and rule execution depend on the cloud. A major advantage is that the IoT hub can also serve as a bridge for cloud-to-cloud integrations, thereby enabling interoperability with vendor-specific ecosystems (e.g., Sonoff via eWeLink Cloud) and extending device support beyond the native protocols, adding value to the growth of the SISS [12, 13].

3.4 Intermediary Edge Gateway at the Edge-based Platform

As shown in Figure 1 d), data reading from the end device is first sent to the local gateway if the device doesn't operate on same protocols as an edge-based platform and the translation is needed (e.g. connecting IKEA Smart bulbs via IKEA Dirigera hub). Otherwise, device connects directly to the platform. An edge-based platform manages connected devices, applies local control logic and performs filtering or data transformation when necessary. An additional layer typically implemented within the edge-based platform, the edge gateway, acts as a translator and forwarder to the cloud platform, driving the progress of the SISS [12, 13]. It bridges local communication with the protocols and formats expected by the cloud (e.g., JSON over HTTP, MQTT, AMQP), while also handling all security requirements such as TLS certificates, keys or tokens toward cloud platform. Another key feature of the edge gateway is its ability to provide offline buffering, ensuring that any data captured during connectivity loss is stored locally and forwarded to the cloud once the connection is restored. In the case of changing cloud IoT platform just translation layer in edge gateway should be changed. In the case of changing smart device only integration of that device in local plaform need to be changed. This way we have flexible and longstanding deployment. An example of such integration in industrial and more professional scenarios is AWS IoT Greengrass [27] edge computing platform, that has more advanced actions executing at the edge, such as running ML models, but however AWS Cloud provides centralized device management, data synchronization and storage, secure authentication, advanced analytics and model distribution and updates. Similar to that, Microsoft Azure IoT Edge [20] shares this concept, where edge runtime is a local software managing edge modules and communication with the cloud. An open-source solution, ThingsBoard IoT Gateway [32], acts as a bridge between end devices and ThingsBoard platform in the cloud, enabling connection for different types of devices and protocols.

In this paper, we present a case study on integrating Home Assistant with Node-RED as an edge gateway, providing connectivity to the cloud IoT platform Data JediX from Ericsson Nikola Tesla [31]. This setup leverages Home Assistant for local device management and automation, while Node-RED ensures communication with the cloud, making the implementation particularly suitable for home automation scenarios. This architecture revolves around several key aspects: ensuring security by protecting data in transit and authenticating components, maintaining privacy by keeping sensitive information local whenever possible, guaranteeing reliability through mechanisms that prevent data loss and handle connectivity issues, and providing flexibility to handle various data formats and integrate heterogeneous systems. Security requires that the edge component supports TLS/SSL encryption and authentication (e.g., certificates, API keys) with secure protocols such as MQTT over TLS or REST with tokens. Also, gateways have their own credentials. Privacy is addressed through a hybrid edge–cloud approach that keeps sensitive data local (e.g., raw images or video) while sending only aggregated or just necessary information to the cloud, with strict role-based access control in Data JediX. Reliability requires mechanisms to mitigate data loss or delayed real-time processing due to connectivity issues; this is partly ensured by store-and-forward at the edge gateway, MQTT persistent sessions, timestamped data for time-series synchronization and redundancy such as dual connectivity (primary Internet with 4G/5G fallback). Finally, flexibility is achieved by handling heterogeneous data formats (JSON, XML, CSV, binary protocols) at the edge, where data can be translated and harmonized; for instance, mapping a Home Assistant entity into a REST payload consumable by platforms like Data JediX. JediX payload is a syntactic data exchange format which doesn't have predefined semantics.

4 Case study: Home Assistant integration with Data Jedi X via Edge Gateway

Home Assistant is an open source IoT platform with the primary application of smart home automation. It allows connecting and controlling different devices in one local network and runs locally on devices such as Raspberry Pi, thus does not depend on cloud computing services, which enables work without an Internet connection and greater control over data. It supports more than 1,000 integrations, including devices and protocols such as MQTT, Zigbee, Z-Wave, Philips Hue, IKEA TRÅDFRI, Google Assistant and Amazon Alexa. The interface is accessible through a web browser or a mobile application and allows viewing the state of the device, managing and creating automations.

Data JediX is an IoT data management platform, developed entirely within Ericsson Nikola Tesla (ENT) for a broad aspect of applications, such as smart city, industry environments, public health and agriculture. It enables collection, transformation, routing and storage of data from any IoT device, as well as sending messages to devices, applications or users. In addition, the system ensures the protection of data and devices from unauthorized access through advanced security mechanisms. Main functionalities include modular data flows, adapter framework for the transformation of input and output data, an information model for data enrichment and the possibility of advanced analytics and building business rules through the Rule Engine. The platform supports work in virtualized environments, containers, and on physical infrastructure. Key functional blocks include:

(1) **Registry** - for hierarchical management of devices, users and applications
(2) **Data Repository** - for secure data storage, independent of the protocol
(3) **Subscription/Notification system** - for managing subscriptions and notifications in real time
(4) **Workflow Engine and Data Transformation module** - for adjusting the flow and data format in real time
(5) **Rule Engine** - for defining automated business rules
(6) **Analytics module** - for basic and advanced data processing
(7) **Intuitive user interface** - for entity introduction, configuration and visualization of data
(8) **Security controls** - including authentication, authorization and login

To register devices on the Data JediX platform, there is a need to establish the organizational structure, which begins with defining the operator (e.g., `SmartEnvironment`), the associated domain (e.g., `SmartUniversity`), the organization (e.g., `FER_org`) and the user entity (e.g., `FER_Departments`). Within this framework, a gateway group (`FER_building_A`) is created to represent the physical

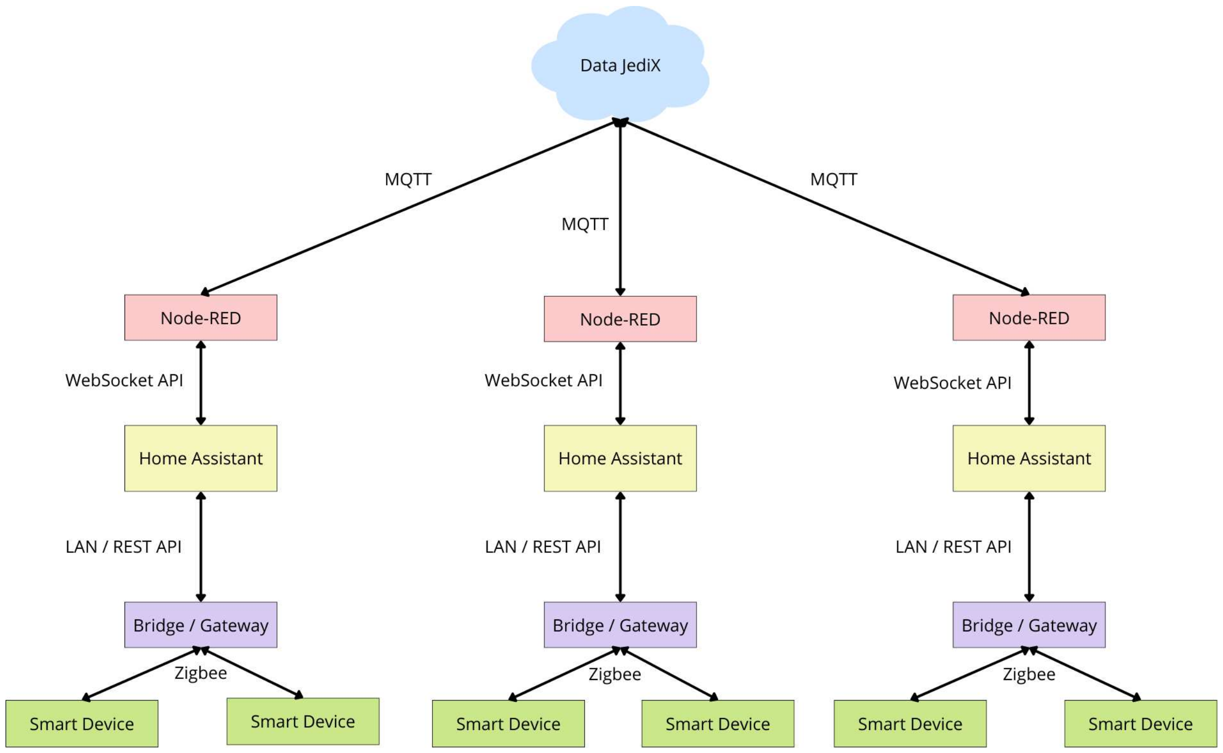


Figure 2: Architecture of the system integrating local and in-the-cloud platform through edge gateway

location in which the devices are deployed. Once the organizational layer is defined, specifications are created to serve as templates for different types of components: the gateway group specification (e.g., UniversityBuilding_GW_GRP), the gateway specification (e.g., HomeAssistant_GW_SPEC), sensor specifications (e.g., HueSmartLight_Spec, HueMotionSensor_Spec), and corresponding resource specifications (e.g., HueSmartLightRes_Spec, HueMotionSensorRes_Spec). Based on these templates, concrete instances are then added. Two gateways are registered (e.g., Home_Assistant_1 and Home_Assistant_2), each linked to the FER_building_A group and the gateway specification. Individual sensors, such as HueGoLamp_1, HueColorLamp_1, and HueMotionSensor_1 for the first gateway, or HueGoLamp_2, HueColorLamp_2, and HueTemperature for the second, are associated with the appropriate gateway and sensor specification. Finally, resources are defined to capture the data of each sensor (e.g., HueMotionSensor_1_Res for the motion sensor HueMotionSensor_1), each connected to its corresponding resource specification. With this hierarchical structure in place, devices become fully registered on the platform and ready for subsequent automation through subscriptions and rules.
Node-RED is an open-source, JavaScript-based flow programming environment, originally developed within IBM and now part of the OpenJS Foundation. It enables easy connection of devices, services and applications through a visual interface. This makes it suitable for applications ranging from smart home automation to industrial control systems. Its low-code nature allows users of varying technical expertise to create system logic by connecting visual nodes representing inputs, conditions, data processing and outputs. Node-RED runs locally on edge devices like Raspberry Pi, on cloud platforms such as AWS or Azure and via containers, leveraging Node.js for efficiency. It supports standard protocols including MQTT, HTTP, WebSocket and TCP, stores flows in JSON format for easy sharing, and offers an online library with ready-made integrations, including Home Assistant, cloud platforms and databases [16].
The architecture of the realized integration is shown in Figure 2. At the local layer, Home Assistant manages Philips Hue devices via the Zigbee Hue Bridge and enables local automation and two-way communication with devices without the need for the Internet. Node-RED utilizes WebSocket API for communicating with the Home Assistant, while it connects Home Assistant and Data JediX using MQTT. A remote MQTT broker on the Data JediX platform enables communication between the local and cloud layers, with Node-RED acting as an MQTT client, ensuring reliable, asynchronous two-way data exchange. Through this common broker, Data JediX centrally registers local instances of Home Assistant, offering unified device management, automation, data storage, analytical reporting and system control across multiple locations.

The functionality of the solution is shown in Figure 3, where a sequence diagram shows in detail the communication flows between system components. The Data JediX platform tracks values coming from local motion sensor. When a true value is detected on one of the sensors, a command is sent to turn on all smart lamps on all connected Home Assistant instances. The command is sent via MQTT to a topic subscribed to by all local systems. Each local system receives the message, recognizes the command and turns on the local smart lamps. To achieve this functionality, a flow was created in the Node-RED. The first part of the flow is about retrieving the state of the motion sensor from the Home Assistant platform. In the initial part of the flow, a node of type `inject` is used, for the purpose of initiating the execution of the data flow at regular time intervals. In this system, the `inject` node is configured to generate a new message every 5 seconds that activates the retrieval of the current state of the motion sensor. The node is configured so that the repeat time option is selected in the Repeat field, with the value set to 5 seconds. The payload and topic fields are left empty because they are not needed for this operation. This node is connected directly to the `api-current-state` node which retrieves the state of the motion sensor `binary_sensor.hue_motion_sensor_1_motion` from the Home Assistant platform. Then a node called `api-current-state` belonging to the group of nodes that come with the `node-red-contrib-home-assistant-websocket` plugin is used, which allows direct integration with the Home Assistant platform via its WebSocket API connection. The `api-current-state` node allows retrieving the current state of any entity in Home Assistant. In the configuration of the node, it is necessary to specify the entity that we want to read. In this case, the motion sensor identifier `binary_sensor.hue_motion_sensor_1_motion` is entered. After this node receives the data from the Home Assistant platform, the message is sent to two nodes. The first of these is the `debug` node, which is used to print data in the sidebar of the Node-RED interface to check the accuracy of the flow execution. The second node is a function node named "DataJediX format". A `Function` node is a general node type in Node-RED that allows logic to be written in JavaScript. In this case, it is used to transform the data format from Home Assistant to the format defined by the Data JediX platform standards. Inside the node, a JSON object is formed with a special structure consisting of an array of `contentNodes`. Each element of that array represents a single piece of data that is sent to the platform. The key part of each element is the `source` object, which must contain the `resource` property because it identifies exactly which resource is sending the data, as shown in Listing 1.

Listing 1: JSON object content inside node Function

```
msg.payload = {
    contentNodes: [
        {
            source: {
                resource: "MotionSensorRes"
            },
            value: msg.payload === "on" ? "true" :
                 "false"
        }
    ]
};
return msg;
```

Without this identifier, the Data JediX platform cannot properly associate the received data with the corresponding device or sensor. In addition to the `source` object, each element also contains a `value` property that carries the actual value of the data. In the specific case of the motion sensor, the value is set to `true` if the sensor is activated, or `false` for all other cases. This transformation ensures that the data matches the expected format and data type that the platform expects. The output from this `Function` node is connected to the `mqtt out` node, which sends a prepared message to the remote MQTT broker. The `mqtt` node is used to publish messages on a specific MQTT topic. In this case, the `digiphy1/in/HueMotionSensor_1` topic is used. Topic indicates the path of data on the Data JediX platform, where `digiphy1` represents the user, `in` is the input channel for data and `HueMotionSensor_1` is the specific name of the sensor. When configuring the `mqtt out` node, it is necessary to define the MQTT broker beforehand. In this case, a remote broker at the address `djx.entlab.hr` and `port 1883` is used. After the node is configured, it is possible to publish messages on topics that we define ourselves. In this way, the actual readings from the motion sensor are sent from Home Assistant to the Data JediX platform in the prescribed format. In addition to sending data to the cloud, the system also enables receiving management commands from the cloud, which come from the Data JediX platform via MQTT. For this, the `mqtt in` node is used, which listens to the `digiphy1/out/datajedix_incoming` topic, where `digiphy1` denotes the user, `out` represents the output path for data from the Data JediX platform, and `datajedix_incoming` is the name of the topic that we arbitrarily chose on the Data JediX platform as the destination for data from the subscription. Messages arriving at this node are simultaneously forwarded to two outputs. The first goes to the `debug` node that monitors incoming data, while the second leads to the `Function` node that checks whether the message contains a command to activate the light on smart lamps, as shown in Listing 2.

Listing 2: Function to check for the presence of a command to activate the light

```
if (
  msg.payload &&
  Array.isArray(msg.payload.contentNodes)
) {
  const effectNode = msg.payload.contentNodes.find(node
       =>
    node.metadata &&
    node.metadata.name === "TurnOnLampEffect" &&
    node.value === true
  );

  if (effectNode) {
    return msg;
  }
}
return null;
```

A simple check is implemented in the `Function` node. If the message contains an array of `contentNodes`, the function searches within that array for an element that has metadata named `TurnOnLampEffect` with the value true. If such an element exists, the message is forwarded, otherwise no action is taken. If the condition in the `Function` node is met, the message is sent to the `api-call-service` node that calls the Home Assistant service `light.toggle`. In the configuration, it is defined that the service `light.toggle` applies to two entities: `light.hue_color_lamp_1` and `light.hue_go_1`. This makes it possible to turn on the `HueColorLamp_1` and `HueGoLamp_1` smart lamps. When configuring this node, it is necessary to enter the name of the *light.toggle* service in the `Service` field and enter the identifiers of all entities in the `Entity ID` field. We do the same when configuring the action node in the second

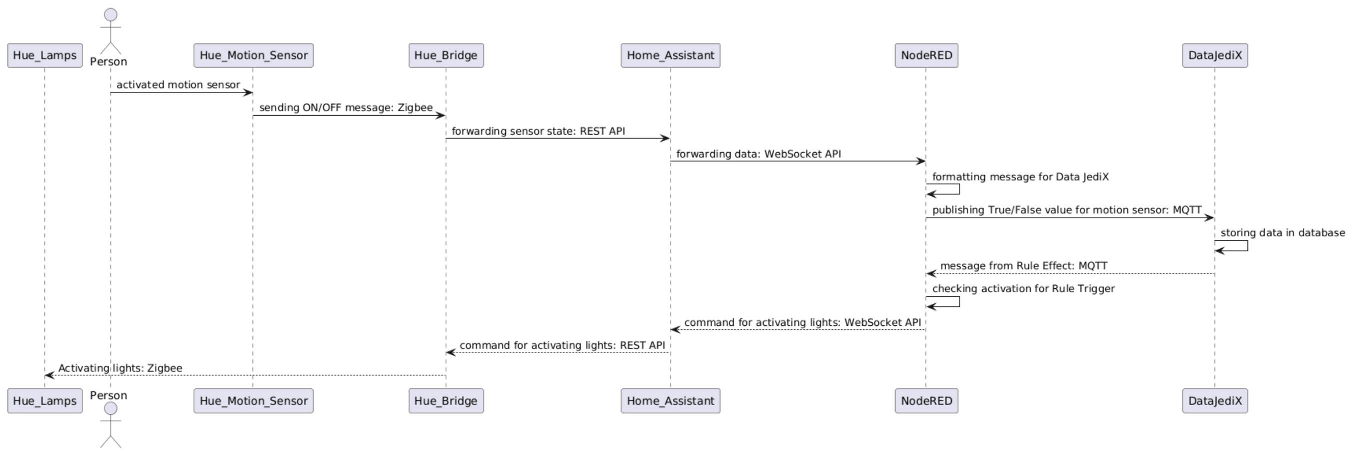

Hue_Lamps
Person
Hue_Motion_Sensor
Hue_Bridge
Home_Assistant
NodeRED
DataJediX
activated motion sensor
sending ON/OFF message: Zigbee
forwarding sensor state: REST API
forwarding data: WebSocket API
formatting message for Data JediX
publishing True/False value for motion sensor: MQTT
storing data in database
message from Rule Effect: MQTT
checking activation for Rule Trigger
command for activating lights: WebSocket API
command for activating lights: REST API
Activating lights: Zigbee

Figure 3: Automation sequence diagram from the perspective of the local environment in which the motion sensor activation occurred

flow for the second instance of Home Assistant. The second configuration allows us to turn on smart lamps that are connected to another instance of Home Assistant. These are the smart lamps `HueGoLamp_2` and `HueColorLamp_2`.

4.1 Local data processing

Before applying format transformation to the incoming data, Node-RED is capable of filtering based on predefined criteria. In the presented case study, "Collect & filter data" `Function` node accesses previously stored device states from the flow context. Incoming values from `msg.payload`, received as strings, are parsed to the float data type. The flow then checks if the value is a valid number. If it is, the value is appended to the stored data array and updated in the flow context. This ensures that only numerically valid sensor data is collected, providing a clean and consistent dataset for subsequent processing. Other than filtering, additional operations like aggregation or computing minimum/maximum values can be performed on the collected data. For example, the "Avg value" `Function` node computes the average temperature every fifteen minutes. It first retrieves the recorded sensor readings from the flow context into a local array, filtering out any non-numeric values. If the array is empty after filtering, the node returns null and no message is forwarded. The remaining numeric values are summed and divided by the array length to calculate the average. This value is then rounded to one decimal place, assigned to `msg.payload` and sent to the next node. After this, the stored values in the flow context are cleared to prepare for the next cycle, as shown in Listing 3.

Listing 3: Filtering the data and calculating the mean value

```
let data = flow.get("values") || [];

data = data.filter(v => typeof v === 'number' &&
     !isNaN(v));

if (data.length === 0) {
  return null;
}

let sum = data.reduce((a, b) => a + b, 0);
let avg = sum / data.length;
avg = parseFloat(avg.toFixed(1));
msg.payload = avg;
flow.set("values", []);

return msg;
```

The application of filtering and data processing operations can significantly reduce network traffic and enhance privacy, as some data remain confined to the edge-based platform. Moreover, it mitigates the risk of potential adversaries capturing raw data, since only aggregated or otherwise processed values are transmitted. This hybrid solution also supports automated actuation, which, when executed locally by Home Assistant, reduces latency and becomes an integral part of the local processing capabilities.

4.2 Interoperability

Interoperability can be understood as the ability of diverse systems, devices and organizations to communicate, exchange data and use it in a consistent and meaningful way. At the technical level, this requires alignment of hardware and software through shared protocols, interfaces and data formats to enable effective information exchange. Syntactic interoperability addresses the structure of data, typically by relying on standardized representations such as XML or JSON, while semantic interoperability ensures that exchanged information carries the same meaning for all parties by adopting common data models or ontologies. In addition, organizational and legal interoperability refer to the policies, agreements and regulatory frameworks that make collaboration across different entities possible. In the IoT domain, interoperability is particularly critical due to the heterogeneity of devices and platforms, which often operate on dissimilar protocols or data formats. Introducing an edge gateway between local systems and the cloud helps bridge these differences, ensuring both technical and syntactic interoperability. In the presented case study, communication between the Home Assistant system and the Data JediX platform relies on the MQTT protocol at the transport layer, where messages are published and received on predefined topics (e.g. `digiphy1/in/...` for input data and `digiphy1/out/...` for output data). The content of each MQTT message must be formatted in JSON format that is compliant with the standards of the Data JediX platform. The key component of this format is the `contentNodes` field, which represents an array of elements, where each element contains an object `source` with the `resource` property that uniquely identifies the data source and the `value` field that carries the actual value of the sensor reading. Additionally, elements can contain a `metadata` field, which is used to transmit control commands and effects (e.g. `TurnOnLampEffect`). In this way, data from the sensor is transformed into syntax of the prescribed JSON format using the `Function` node in Node-RED and sent to the Data JediX platform, while commands from the platform come in the same format and are forwarded to Home Assistant services for execution.

5 Conclusion

Analysis of existing connectivity strategies indicates that a two-tier architecture with an Intermediary Edge Gateway provides an effective balance between local autonomy and global scalability. This approach enables cloud-adapted payload transformation, while enhancing data throughput efficiency with filtering and aggregation. A case study involving the open-source edge platform Home Assistant, the commercial Data JediX cloud platform, and Node-RED as an intermediary demonstrates that the proposed architecture enables scalable, modular management of heterogeneous IoT devices. A key advantage of the described IoT integration lies in its durability: the edge (local) platform can remain fixed, while the cloud provider can be changed or updated over time. This approach ensures stability at the edge for connected devices while providing flexibility to switch or upgrade cloud services as needed. By combining local preprocessing with centralized processing, this hybrid solution enhances interoperability, reduces reliance on network connectivity, and improves operational efficiency, providing a solid foundation for persistent IoT solutions. However, the presented case study serves primarily as an initial proof of concept. Building upon this foundation, future work could include a comprehensive empirical evaluation encompassing key performance metrics such as latency, throughput, scalability, expected network traffic reduction, and CPU utilization on the gateway. Furthermore, extending the system to integrate with other cloud-based IoT platforms would enable assessment of its generalizability and interoperability across diverse deployment environments.

6 Acknowledgments

This research has been funded by the European Union – NextGenerationEU, under the project XR Communication and Interaction Through a Dynamically Updated Digital Twin of a Smart Space – DIGIPHY, grant number NPOO.C3.2.R3-I1.04.0070. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Commission. Neither the EU nor the EC can be held responsible for them.

References

[1] A.A.H. Al-Shateri et al. 2024. Luna: A Benchmark Project in the Convergence of Artificial Intelligence and Internet of Things for Home Automation. In *2024 IEEE International Conference on Advanced Telecommunication and Networking Technologies (ATNT)*, Vol. 1. 1–4. doi:10.1109/ATNT61688.2024.10719226

[2] F.C. Andriulo et al. 2024. Edge Computing and Cloud Computing for Internet of Things: A Review. *Informatics* 11, 4 (2024). https://www.mdpi.com/2227-9709/11/4/71

[3] N.A. Angel et al. 2022. Recent Advances in Evolving Computing Paradigms: Cloud, Edge, and Fog Technologies. *Sensors* 22, 1 (2022). https://www.mdpi.com/1424-8220/22/1/196

[4] Home Assistant. 2025. Home Assistant Integrations - Amazon Web Services (AWS). https://www.home-assistant.io/integrations/aws/. Accessed: 2025-09-09.

[5] Home Assistant. 2025. Home Assistant Integrations - Azure Event Hub. https://www.home-assistant.io/integrations/azure_event_hub/. Accessed: 2025-09-09.

[6] Home Assistant. 2025. Home Assistant Integrations - Documentation. https://www.home-assistant.io/docs/. Accessed: 2025-09-09.

[7] Home Assistant. 2025. Home Assistant Integrations - Google Cloud. https://www.home-assistant.io/integrations/google_cloud/. Accessed: 2025-09-09.

[8] Home Assistant. 2025. Home Assistant Integrations - IBM Watson IoT Platform. https://www.home-assistant.io/integrations/watson_iot/. Accessed: 2025-09-09.

[9] M. Bauer. 2022. FIWARE: Standard-based Open Source Components for Cross-Domain IoT Platforms. In *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*. 1–6. doi:10.1109/WF-IoT54382.2022.10152259

[10] F. Cirillo et al. 2019. A Standard-Based Open Source IoT Platform: FIWARE. *IEEE Internet of Things Magazine* 2, 3 (2019), 12–18. doi:10.1109/IOTM.0001.1800022

[11] N. Das and M. Shakil. 2025. Design and Implementation of an IoT based Energy-Efficient User-Friendly Smart Home Automation System for Energy Savings and Maximizing Comfort. 1–5. doi:10.1109/ECCE64574.2025.11013781

[12] M.S.E. de la Parte et al. 2023. Breaking Down IoT Silos: Semantic Interoperability Support System for the Internet of Things. In *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. 1–7. doi:10.1109/ICECCME57830.2023.10253024

[13] M.S.E. de la Parte et al. 2025. SISS: Semantic Interoperability Support System for the Internet of Things. *IEEE Internet of Things Journal* 12, 16, 33769–33791. doi:10.1109/JIOT.2025.3577776

[14] O. Debauche et al. 2022. A New Edge Computing Architecture for IoT and Multimedia Data Management. *Information* 13, 2 (2022). doi:10.3390/info13020089

[15] Eclipse. 2025. Eclipse Kura project. https://projects.eclipse.org/projects/iot.kura. Accessed: 2025-09-15.

[16] Nijhof F. 2025. Home Assistant Community Add-on: Node-RED. https://community.home-assistant.io/t/home-assistant-community-add-on-node-red/55023/1. Accessed: 2025-09-09.

[17] K. Habib et al. 2022. An Aggregated Data Integration Approach to the Web and Cloud Platforms through a Modular REST-Based OPC UA Middleware. *Sensors (Basel)* 22, 5 (2022).

[18] G. Hatzivasilis et al. 2019. Secure Semantic Interoperability for IoT Applications with Linked Data. In *2019 IEEE Global Communications Conference (GLOBECOM)*. 1–6. doi:10.1109/GLOBECOM38437.2019.9013147

[19] P. Schoutsen Home Asisstant. 2025. Classifying the Internet of Things. https://www.home-assistant.io/blog/2016/02/12/classifying-the-internet-of-things/#classifiers. Accessed: 2025-09-09.

[20] Microsoft. 2025. What is Azure IoT Edge. https://learn.microsoft.com/en-us/azure/iot-edge/about-iot-edge. Accessed: 2025-09-09.

[21] openHAB. 2025. Add-on Reference, Bindings. https://www.openhab.org/addons/. Accessed: 2025-09-09.

[22] openHAB. 2025. openHAB - Documentation. https://www.openhab.org/docs/. Accessed: 2025-09-09.

[23] V.N. Pham et al. 2021. Efficient Solution for Large-Scale IoT Applications with Proactive Edge-Cloud Publish/Subscribe Brokers Clustering. *Sensors* 21, 24 (2021).

[24] I. Podnar Žarko et al. 2017. Towards an IoT framework for Semantic and Organizational Interoperability. In *2017 Global Internet of Things Summit (GIoTS)*. 1–6. doi:10.1109/GIOTS.2017.8016253

[25] I. Podnar Žarko et al. 2019. The symbIoTe Solution for Semantic and Syntactic Interoperability of Cloud-based IoT Platforms. In *2019 Global IoT Summit (GIoTS)*. 1–6. doi:10.1109/GIOTS.2019.8766420

[26] P. Savaridass et al. 2025. A Multi-Protocol IoT-Enabled Touch Control Switch for Energy-Efficient Smart Home Automation. In *2025 Second International Conference on Cognitive Robotics and Intelligent Systems (ICC - ROBINS)*. 133–139. doi:10.1109/ICC-ROBINS64345.2025.11086234

[27] Amazon Web Services. 2025. AWS IoT Greengrass, Developer Guide, Version 2. https://docs.aws.amazon.com/pdfs/greengrass/v2/developerguide/greengrass-v2-developer-guide.pdf#what-is-iot-greengrass. Accessed: 2025-09-09.

[28] J. Shiraishi et al. 2024. Coexistence of Push Wireless Access with Pull Communication for Content-based Wake-up Radios. In *GLOBECOM 2024 - 2024 IEEE Global Communications Conference*. 4836–4841. doi:10.1109/GLOBECOM52923.2024.10901717

[29] I. V. Sita and B. David. 2024. Development and Implementation of a Comprehensive Smart Home Automation System Using Home Assistant and IoT Devices. In

2024 4th International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME). 1–7. doi:10.1109/ICECCME62383.2024.10796669

[30] E. O. Sodiya et al. 2024. Current State and Prospects of Edge Computing within the Internet of Things (IoT) Ecosystem. *International Journal of Scientific Research and Analysis* 11, 1 (2024), 1863–1873. https://ijsra.net/sites/default/files/IJSRA-2024-0287.pdf

[31] Ericsson Nikola Tesla. 2025. Data Management Platform. https://ericssonnikolatesla.com/en/digital-society/platforms/data-management-platform/. Accessed: 2025-09-09.

[32] ThingsBoard. 2025. What is ThingsBoard IoT Gateway? https://thingsboard.io/docs/iot-gateway/what-is-iot-gateway/. Accessed: 2025-09-09.

[33] A. Tolcha et al. 2021. Towards Interoperability of Entity-Based and Event-Based IoT Platforms: The Case of NGSI and EPCIS Standards. *IEEE Access* 9, 49868–49880. doi:10.1109/ACCESS.2021.3069194

From Noise to Knowledge: A Comparative Study of Acoustic Anomaly Detection Models in Pumped-storage Hydropower Plants

Karim Khamaisi
University of St. Gallen
Dornbirn, Austria
karim.khamaisi@unisg.ch

Nicolas Keller
University of St. Gallen
St. Gallen, Switzerland
nicolas.keller@student.unisg.ch

Stefan Krummenacher
University of St. Gallen
St. Gallen, Switzerland
stefan.krummenacher@student.unisg.ch

Valentin Huber
University of St. Gallen
St. Gallen, Switzerland
valentin.huber@student.unisg.ch

Bernhard Fässler
illwerke vkw AG
Bregenz, Austria
bernhard.faessler@illwerkevkw.at

Bruno Rodrigues
University of St. Gallen
Dornbirn, Austria
bruno.rodrigues@unisg.ch

Abstract

In the context of industrial factories and energy producers, unplanned outages are highly costly and difficult to service. However, existing acoustic-anomaly detection studies largely rely on generic industrial or synthetic datasets, with few focused on hydropower plants due to limited access. This paper presents a comparative analysis of acoustic-based anomaly detection methods, as a way to improve predictive maintenance in hydropower plants. We address key challenges in the acoustic preprocessing under highly noisy conditions before extracting time- and frequency-domain features. Then, we benchmark three machine learning models: LSTM AE, K-Means, and OC-SVM, which are tested on two real-world datasets from the Rodundwerk II pumped-storage plant in Austria, one with induced anomalies and one with real-world conditions. The One-Class SVM achieved the best trade-off of accuracy (ROC AUC 0.997–0.998) and minimal training time, while the LSTM autoencoder delivered strong detection (ROC AUC 0.997-0.999) at the expense of higher computational cost.

CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Applied computing** → *Industry and manufacturing*.

Keywords

Industrial Machinery, Acoustic Anomaly Detection, Predictive Maintenance, Machine learning

1 Introduction

Maintenance is a critical function in industrial environments due to the significant operational and financial impacts associated with equipment failure. Recent estimates highlight that unplanned downtime costs industries billions annually, with individual facilities losing millions per event [27]. Such unexpected interruptions are particularly consequential in energy production facilities like pumped-storage hydropower plants, where operational continuity is essential to maintain grid stability and minimize economic impacts.

Hydropower plant equipment, including turbines and generators, is central to plant operations. Failures in these mechanical parts can result in costly downtime. Downtime costs are highly variable, primarily depending on fluctuating market energy prices. For instance, the Rodundwerk II pumped-storage (*cf.* Section 2) hydropower plant in Vorarlberg, Austria, has a full-load capacity of 295 MW, translating to 295 MWh of electricity production per hour. With a wholesale price of 97.30 Euro/MWh [9] (as of February, 2025), downtime can cost over 28,000 Euros per hour [12]. Moreover, since pumped-storage hydropower plants typically function as peak-load power plants, selling electricity predominantly at times of high market prices, actual downtime costs can significantly exceed this estimate.

Predictive maintenance (PdM) proactively anticipates equipment failures, enabling maintenance timing that avoids both unnecessary downtime and catastrophic failures [21, 32]. Among various PdM approaches, acoustic-based anomaly detection stands out for its non-intrusive nature, cost-effectiveness, and sensitivity to early-stage mechanical defects [7, 13, 15]. Acoustic signals carry vital diagnostic information, reflecting subtle deviations in machinery operations that precede potential failures. However, acoustic-anomaly detection methods often face challenges in industrial environments characterized by high noise levels, complex acoustic patterns, and limited labeled anomaly data [17, 28].

Although acoustic anomaly detection has been extensively studied, existing literature predominantly relies on generic industrial or synthetic datasets [2, 7, 8, 24], with limited representation from actual hydropower plant operations. This gap exists mostly due to restricted access to critical infrastructure, such as pumped-storage hydropower facilities, whose acoustic characteristics significantly differ from standard industrial machinery. Consequently, existing models [15, 17, 28] trained on generic data often fail to generalize effectively to the acoustic conditions and anomalies characteristic of hydropower plants.

We provide systematic comparative analysis of acoustic anomaly detection methods explicitly applied to a pumped-storage hydropower plant, Rodundwerk II in Austria managed and operated by illwerke vkw AG [12]. **Contributions** are summarized as follows:

- We present an acoustic preprocessing pipeline tailored to mitigate noise and handle the challenging acoustic environment of hydropower plants.
- Benchmark three prominent machine learning models: LSTM Autoencoder (LSTM AE), K-Means clustering, and One-Class SVM (OC-SVM), using two distinct real-world datasets

Table 1: Selected Related Work on Acoustic Anomaly-detection

Study	Domain / Data	Feature Rep.	Model(s)	Key Insight
Müller et al. (2020) [16]	Factory machinery	MelSpec + CNN feats	IF, GMM, OC-SVM, AE	CNN features beat CAE in noise
Meire et al. (2019) [20]	Industrial bench setup	MFCC, MelSpec	OC-SVM, AE	HW-friendly AEs for edge devices
Bayram et al. (2021) [2]	Chemical process plant	MelSpec	Conv-LSTM-AE, CAE	Sequential AEs allow real-time
Ferraro et al (2023) [11]	Multi-factory lines	MelSpec	LSTM-AE, CNN-AE	Sliding-window edge inference
Coelho et al. (2022) [4]	Factory & vehicle	MFEC, MelSpec	Dense/CNN/LSTM AEs	Depth vs. noise-robustness study
Duman et al. (2020) [8]	Industrial plant	MelSpec	CAE	Lightweight CAE for on-site use
Ahn et al. (2021) [1]	Vehicle acoustics	Raw audio	SVM, K-Means, CNN	Multi-mic array boosts recall
Tagawa et al. (2021) [28]	Rotating machinery	Spectral coeff.	SVM, Auto-corr.	Spectral coeff. key for bearings
Purohit et al. (2019) [24]	MIMII (public)	Log-MelSpec	VAE, AE	Widely used open benchmark

acquired from Rodundwerk II: one with induced anomalies under controlled conditions, and another featuring naturally occurring operational anomalies.

- We discuss strengths and weaknesses of each model, providing guidelines for practitioners on selecting appropriate methods based on operational constraints.

The remainder of this paper is organized as follows: Section 2 reviews existing literature on acoustic anomaly detection, focusing on both traditional signal processing and machine learning approaches. Section 3 presents the proposed framework, detailing its preprocessing pipeline and model architectures. Section 4 discusses experimental results and provides a comparative evaluation real-world datasets. Finally, Section 5 offers concluding remarks and outlines directions for future research.

2 Background and Related Work

Pumped storage hydropower plants (PSHP). Traditional hydropower plants generate electricity using water stored behind a dam, flowing downstream to drive turbines [14]. In contrast, PSHPs utilize mountainous topography with two water reservoirs [22]. During low demand, water is pumped from the lower reservoir to the upper reservoir using surplus energy; during high demand, stored water is released to generate electricity.

Unlike traditional hydropower plants, PSHPs require turbines to operate in dual mode (generation and pumping), leading to frequent operational changes that increase stresses, vibrations, and dynamic loads. These factors accelerate turbine fatigue and wear, causing issues like pump cavitation [31]. Anomaly detection facilitates early fault detection to prevent costly downtime [7, 13], with acoustic-based methods leveraging non-intrusive signals to capture subtle mechanical deviations prior to critical failures [17, 28].

Feature extraction techniques. Robust feature extraction is essential for accurately representing acoustic signals. Traditional methods like Short-Time Fourier Transform (STFT) and Mel-scale transformations convert raw audio signals into time-frequency spectrograms [2, 32]. Mel-spectrograms are widely adopted due to their perceptual alignment with human auditory systems [20], while Mel-Frequency Cepstral Coefficients (MFCCs) capture critical temporal and spectral characteristics that enhance anomaly detection [2, 4].

Machine learning models. Acoustic anomaly detection typically employs machine learning (ML) methods, broadly categorized into unsupervised and deep learning approaches. Density-based models like Local Outlier Factor (LOF)identify anomalies based on local data densities without extensive labeled data [3, 29]. Clustering methods like K-Means offer computational simplicity but struggle with overlapping acoustic patterns [1]. Deep learning methods like Autoencoders (AEs) detect anomalies through reconstruction errors, proving effective in complex environments [8, 11], though they require substantial computational resources often unavailable in hydropower plants.

Table 1 provides a concise comparative overview of recent relevant works, highlighting feature extraction methods, ML models used, and key objectives. Existing work relies mainly on generic or synthetic datasets, with few incorporating real-world data from specific applications such as hydroelectric plants [2, 24]. In terms of feature extraction, Mel-spectrograms are the most commonly used technique [2, 4, 8, 11, 16, 20], followed by MFCCs and STFT [20]. This highlights the importance and effectiveness of converting raw audio signals into meaningful inputs for ML models.

The most used ML models are AEs, which are highly adaptable to specific audio signals and effectively detect abnormal sounds deviating from learned baselines. Various AE architectures appear across studies: [4] explores three different architectures, [11] compares LSTM-AE and CNN-AE models, while [2] utilizes Conv-LSTMAE and CAE. This diversity highlights standardization limitations, as no single model consistently outperforms others across all scenarios.

3 Proposed Method

Figure 2 illustrates the proposed acoustic anomaly detection pipeline. The pipeline addresses the challenges inherent in handling acoustic data from PSHP hydropower plants. It consists of four main stages, which are detailed in next subsections: Data Acquisition, Preprocessing, Model Benchmarking, and Evaluation Metrics.

The design of this pipeline reflects key considerations for handling real-world industrial acoustic-data, including noise resilience, efficient anomaly representation, and scalable deployment. The pipeline comprises four main stages:

(1) **Data Acquisition**: audio recordings of machine operations in controlled and real-world environments.
(2) **Preprocessing**: transformation of raw audio into normalized Mel-spectrograms.
(3) **Model Training**: benchmarking and evaluation of five selected models.

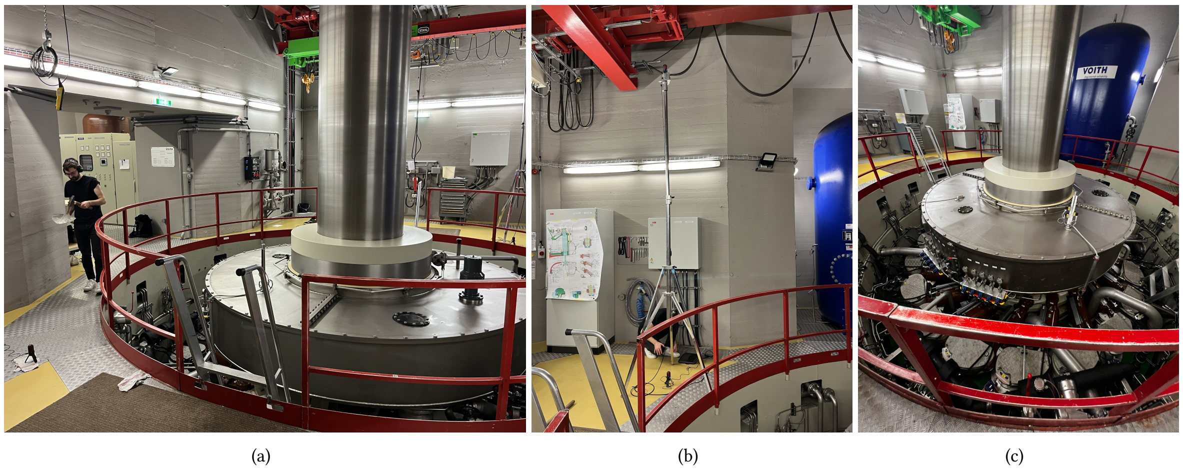


Figure 1: Data Acquisition in PSHP Rodundwerk II: (a) audio recordings with induced anomalies (hammer and a shovel). (b) microphone stands in 4 corners capturing airborne audio. (c) shaft of the reversible francis-pump turbine; microphone stands were installed in each corner of the room.

(4) **Evaluation Metrics**: identification of anomalies using reconstruction error and comparing with the benchmarks.

3.1 Data Acquisition

Two distinct datasets were produced on-site to benchmark models for the acoustic anomaly detection. This was achieved by recording the machinery in real operating modes and with induced anomalies, splitting them into separate *.wav* files, used for data preprocessing.

The dataset with **induced anomalies** was recorded on-site to help in the first stage of fine-tuning models (cf Figure 1 - a). We used a hammer striking a shovel to resemble real-world structural issues yet remain safe and repeatable. These synthetic knocks allow replication of realistic fault phenomena and thus form the basis for testing the framework under near–real-world conditions while preserving consistency in labeling.

The **real dataset** was collected from an operational pumped storage hydropower plant under strict operational constraints that inherently limit data collection scope (in which the authors are thankful to the Rodundwerk's technical team). Access to such facilities is restricted due to safety regulations, security protocols, and the need for coordination with plant operators.

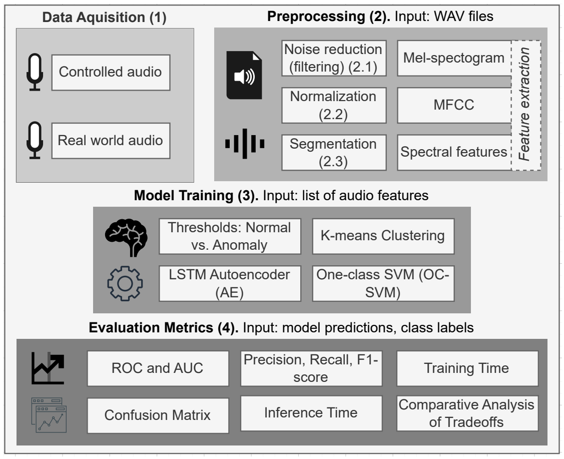

Data Aquisition (1)
Controlled audio
Real world audio
Preprocessing (2). Input: WAV files
Noise reduction (filtering) (2.1)
Normalization (2.2)
Segmentation (2.3)
Mel-spectogram
MFCC
Spectral features
Feature extraction
Model Training (3). Input: list of audio features
Thresholds: Normal vs. Anomaly
K-means Clustering
LSTM Autoencoder (AE)
One-class SVM (OC-SVM)
Evaluation Metrics (4). Input: model predictions, class labels
ROC and AUC
Precision, Recall, F1-score
Training Time
Confusion Matrix
Inference Time
Comparative Analysis of Tradeoffs

Figure 2: Overview of the acoustic anomaly detection framework

3.2 Preprocessing

Prior to modelling, raw audio signals must be converted into representations that are both stable and discriminative in the context of acoustic anomaly detection. We use a Short-Time Fourier Transform (STFT) followed by a Mel-spectrogram conversion and a frame-based segmentation procedure. These steps are essential to transform the original time domain signals into a form more suited to machine learning methods. In addition, we used a combination of **noise reduction** to provide cleaner audio features and reduce the influence of noisy backgrounds; **normalization** using root mean square through average amplitude levels to ensure consistency while preserving feature stability; and **segmentation** splitting audio recordings into smaller, overlapping frames before being input to the ML model, facilitating feature extraction and maintaining balanced datasets [26].

To extract audio features, we first apply a standard discrete *Short-Time Fourier Transform (STFT)* [5, 19, 23] from the *librosa* library with a 1024-sample window and a hop length of 512. By decomposing the audio into short, overlapping time segments, we obtain local frequency-domain information, enabling the detection of short-duration events (*e.g.,* knocks or impacts). This time-frequency representation is critical for capturing the subtle temporal patterns that characterize anomalies in industrial or mechanical systems. Mathematically, the STFT is defined in Equation 1, where we sum over windowed signal frames and apply the discrete Fourier transform.

$$\mathrm{STFT}(n,k) = \sum_{m=0}^{L-1} x(n+m)\, w(m)\, e^{-j\,2\pi\,\frac{k}{N}\,m} \tag{1}$$

From the STFT magnitude, we derive a 128-band *Mel-spectrogram* [6, 18, 19]. The Mel scale aligns more closely with human auditory perception, ensuring that frequency regions vital to detecting anomalies (*e.g.*, sudden high-frequency bursts) are not overshadowed by less relevant areas of the spectrum. By compressing the frequency axis in this perceptual manner, the model is better able to focus on critical frequency components that differentiate normal from abnormal operations.

$$M(m,n) = \sum_{k=0}^{K-1} H_m(k)\, |STFT(n,k)|^2 \tag{2}$$

In Equation 2, each Mel bin is computed by summing the STFT power according to a triangular filter bank H_m.

$$M_{\mathrm{DB}}(m,n) \;=\; 10\,\log_{10}\!\Big(\frac{M(m,n)}{\max_{\mathrm{m,n}}(\mathrm{M(m,n)})}\Big) \tag{3}$$

The Mel-spectrogram is converted to decibels using Equation 3, normalizing each cell by the global maximum and applying $10\log_{10}$. Then, each Mel-spectrogram is min-max normalized [25] to a $[0,1]$ range as shown in Equation 4, which subtracts the minimum and divides by the overall range:

$$x_{\mathrm{norm}} = \frac{x - \min(X)}{\max(X) - \min(X)} \tag{4}$$

This standardization step makes training more robust by reducing amplitude-related variability and ensuring uniform feature scales across the dataset. As a result, the model is not biased by unusually loud or soft recordings, improving generalization to new, unseen data.

```python
time_per_frame = 0.6
hop_ratio      = 0.2
hop_length     = 512

def generate_mel_spectrogram(audio_path):
    audio, sr = librosa.load(audio_path, sr=None)
    stft = librosa.stft(audio, n_fft=1024, hop_length=
        ↪ hop_length)
    mel  = librosa.feature.melspectrogram(S=np.abs(
        ↪ stft)**2, sr=sr, n_mels=128)
    mel_db = librosa.power_to_db(mel, ref=np.max)
    mel_db_norm = (mel_db - mel_db.min()) / (mel_db.
        ↪ max() - mel_db.min())
    return mel_db_norm, sr
```

Listing 1: Code to Generate and Normalize Mel-Spectrograms

The Python code in Listing 1 illustrates how the first steps are implemented in practice, including loading the audio data, applying an STFT, generating and normalizing Mel-spectrograms. After computing the Mel-spectrogram, we segment each spectrogram into overlapping frames. In our setup:

- Windowing: A frame length of 0.512 s is chosen to capture transient acoustic events while retaining sufficient frequency resolution.
- Hop Ratio: A 20 % overlap is maintained between consecutive frames (*i.e.,* a hop ratio of 0.2). This partial overlap preserves temporal continuity and guards against losing information at frame boundaries—particularly important for detecting short, impulsive anomalies.

By transforming each spectrogram into a series of frames, we produce localized snapshots of the acoustic signal. These frames serve as input units for both training and evaluation, making the subsequent anomaly detection task more fine-grained and sensitive to local transient events.

```python
def generate_frames(mel_spectrogram, frame_size,
    ↪ hop_size):

    num_frames = (mel_spectrogram.shape[1] -
        ↪ frame_size) // hop_size + 1
    frames = np.zeros((num_frames, mel_spectrogram.
        ↪ shape[0], frame_size))
    for i in range(num_frames):
        start = i * hop_size
        frames[i] = mel_spectrogram[:, start:start +
            ↪ frame_size]
    return frames
```

Listing 2: Code to Generate Overlapping Frames from a Mel-Spectrogram

The Python code in Listing 2 illustrates how the generation of frames is implemented in practice. Each set of frames is then saved in `.npy` format for subsequent training and evaluation. By separating the raw audio loading from the downstream tasks, we ensure a clear workflow in which all models operate on the same preprocessed feature set.

3.3 Anomaly Detection Models

We selected three machine learning models due to their distinct methodological strengths, computational efficiency, and suitability to handle the complexities inherent to acoustic anomaly detection in hydropower:

- **K-Means clustering**: has a minimal computational overhead and intuitive clustering based approach [1], making it suitable for real-time applications with limited computational resources.
- **Long Short-Term Memory Autoencoder (LSTM AE)**: a deep learning model capable of capturing temporal dependencies in sequential acoustic signals, excelling at identifying deviations from normal behaviors, as can be seen in [10] where accuracy rates of 99% were achieved when detecting anomalies.
- **One-Class Support Vector Machine (OC-SVM)**: a model that learns a tight decision boundary around normal (non-anomalous) data, flagging deviations from the decision boundary as anomalies [16, 20].

In addition, we compute each method's outlier score on the validation set (normal-only) to pick a percentile-based **anomaly threshold**. We evaluated multiple thresholds between the 5th and 95th percentiles, and selected the ones that maximizes the F1-score [30].

4 Experimental Evaluation

This Section presents results of the proposed acoustic anomaly detection framework, evaluated on audio data collected with synthetic, induced anomalies, and in real operation mode. All audio was captured with a *Samson Meteor MIC* at a 16 kHz sampling rate and stored in `.wav` format for consistency. We examine three ML models representing key industrial trade-offs: fast but simple methods, balanced approaches, and high-performance but resource-heavy solutions.

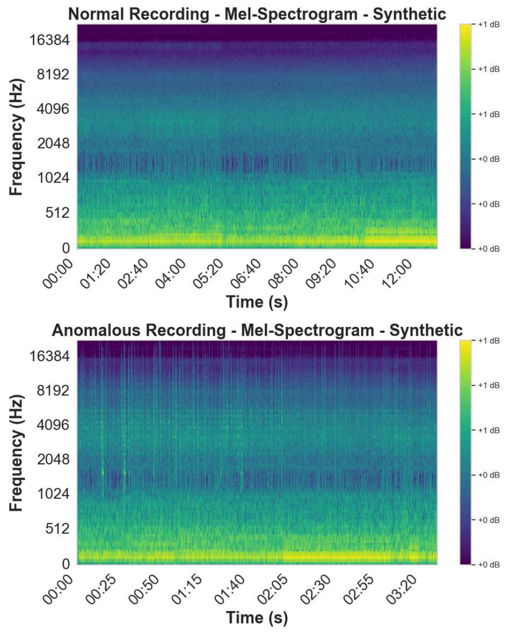


Figure 3: Mel-spectrograms for synthetic dataset: normal (top) vs. anomalous (bottom).

4.1 Induced Synthetic Anomalies

The first dataset was recorded in an operational industrial machine and comprises:

- **Normal operation (12 minutes 46 seconds)**: Equipment functioning under routine conditions, reflecting regular machine activity.
- **Induced events (3 minutes 29 seconds)**: Artificial mechanical faults introduced via controlled impacts from a hammer striking a shovel, selected to resemble real-world structural issues yet remain safe and repeatable.

These synthetic knocks allow replication of realistic fault phenomena and thus form the basis for testing the framework under near–real-world conditions while preserving consistency in labeling.

Evaluation metrics. Table 2 presents the **evaluation metrics** for the ML models on the synthetic anomaly dataset with focus on the K-Means, OC-SVM, and LSTM-AE. All models achieved near-perfect results, as reflected in the ROC AUC, precision, recall, and F1-scores. The **confusion matrices** in Figure 4 further highlight the strong classification performance. Overall, the LSTM-AE provides the most balanced outcome, maintaining both a low false positive rate (3.0%) and high anomaly recall (99.8%), making it well suited for deployment where precision and trust in anomaly alarms are essential. However, K-Means offers the fastest training and inference times, with performance metrics nearly matching those of LSTM-AE, and achieves the lowest overall misclassification count.

The **Mel-spectrograms** of the synthetic anomaly and normal datasets are presented in Figure 3. The normal recording (top) shows smooth, constant energy distribution below 2000 HZ. The anomalous Mel-spectrogram (bottom) reveals multiple energy bursts with vertical stripes representing induced anomalies. It shows short high-energy bands in a high frequency range of above 2000 HZ, which correspond to the knocking sounds caused by a hammer hitting a shovel. Therefore, the normal operation noise is clearly visually distinguishable from the anomalous sound.

Figure 5 shows the **MFCC** coefficients for both recordings. The the normal recording (top) exhibits relatively uniform spectral patterns suggesting a stable envelope, while the anomalous recording (bottom) reveals sudden blanks in coefficients over short intervals, especially in coefficient 0 representing average log-energy. However, spectral patterns remain fairly similar between recordings, likely due to hydroelectric plant background noise.

Figure 6 displays the **FFT Amplitude spectrum**, revealing amplitude distribution across frequencies. The normal recording (top) shows a concentrated spectral profile in the lower frequencies, while the anomalous recording (bottom) exhibits wider energy distribution with distinct amplitude spikes in higher frequencies, corresponding to induced anomalies.

Table 2: Evaluation metrics for the synthetic anomaly dataset.

Method	Train Time (s)	ROC AUC	Precision	Recall	F1-Score	Inference Time (s)
K-Means	0.3700	0.9974	0.9825	0.9989	0.9906	0.0275
OC-SVM	2.8284	0.9977	0.9615	0.9994	0.9801	2.1469
LSTM-AE	32.8777	0.9995	0.9814	0.9972	0.9893	0.4765

4.2 Real-world Operation

The second dataset was recorded in an operational settings and comprises:

- **Normal operation (59 minutes 53 seconds)**: equipment functioning under routine conditions, reflecting regular machine activity.
- **Transient acoustic event (5 seconds within 59 minutes)**: detected during the transition phase, *i.e.,* electricity generation to water pump.

According to operational insights from plant personnel, acoustic anomalies typically occur during specific operational transitions: startup sequences, mode switching between pumping and generation phases, and load adjustment periods. These transient events are **not** indicative of equipment failure but rather reflect the dynamic behavior of turbine systems designed to operate in dual modes. Also, they are brief by nature, as prolonged anomalies would indicate actual equipment malfunction or damage rather than normal operational variations. The short duration of real anomalies (5 seconds within 59 minutes recording) and the constrained dataset size reflect this operational reality where brief acoustic deviations are characteristic of healthy industrial systems, combined with the practical challenges of extended data collection in critical infrastructure environments.

The **Mel-spectrogram** of the real anomaly dataset as well as the normal and anomalous frames are shown in Figure 7. The operating modes are clearly visible through pronounced energy shifts, but these do not necessarily correspond to anomalies since normal mode transitions are prolonged with uniform frequency distribution. Conversely, the anomalous recording (bottom) exhibits inconsistent frequency changes without apparent pattern throughout the recording.

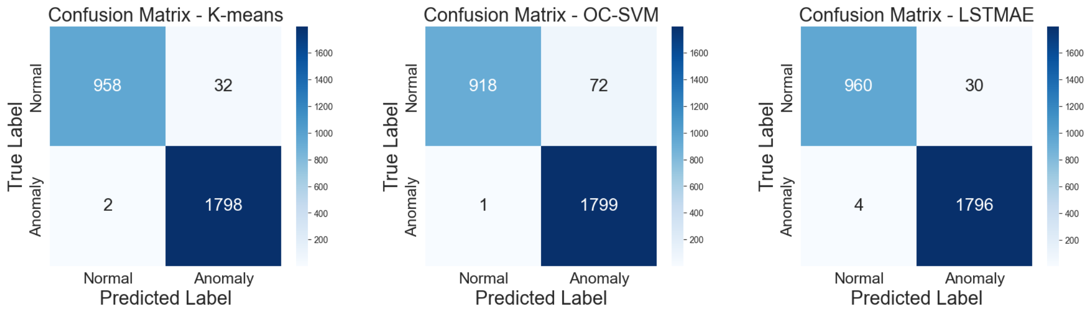


Figure 4: Confusion matrices: Induced (Synthetic) Anomalies Dataset

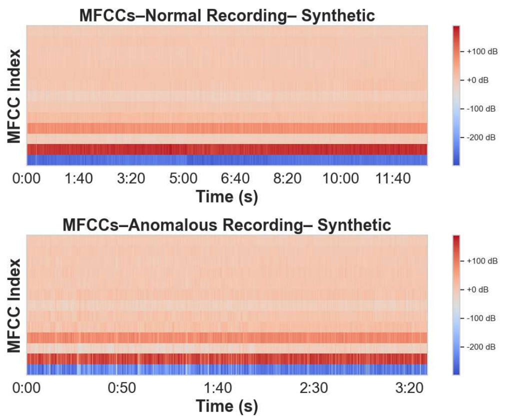


Figure 5: Comparison of Anomalous and Normal Recordings MFCCs

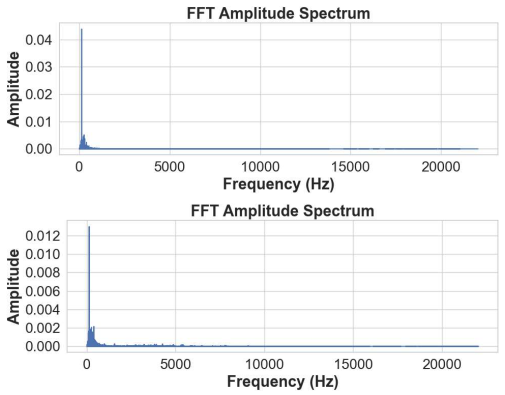


Figure 6: Comparison of Anomalous and Normal Recordings FFT Amplitudes

The previous observations can also be seen in Figure 9, which shows the **MFCCs** of both recordings. The normal MFCC (top) shows consistent color bands in most coefficients in the normal recording indicating a stable spectral envelope. Apart from the evident the changes in operating modes in coefficient 0, each coefficient contains a stable spectral envelope. In contrast, the anomalous recording (bottom) exhibits higher variability in the MFCC coefficients. The coefficients 1-12 fluctuate more intensely during anomalies, showing unstable spectral patterns. Additionally, coefficient 0 shows a lower log-energy suggesting reduced energy in abnormal sounds.

Lastly, Figure 10 shows a comparison of the FFT Amplitude spectrum of both recordings. The amplitude and frequency distribution of the anomalous FFT (bottom) is significantly broader compared to the normal recording (top). The frequency distribution for the normal is much more compact, representing normal operating conditions. Furthermore, the amplitude peak in the normal recording is 0.010dB compared to a higher peak in the anomalous recording

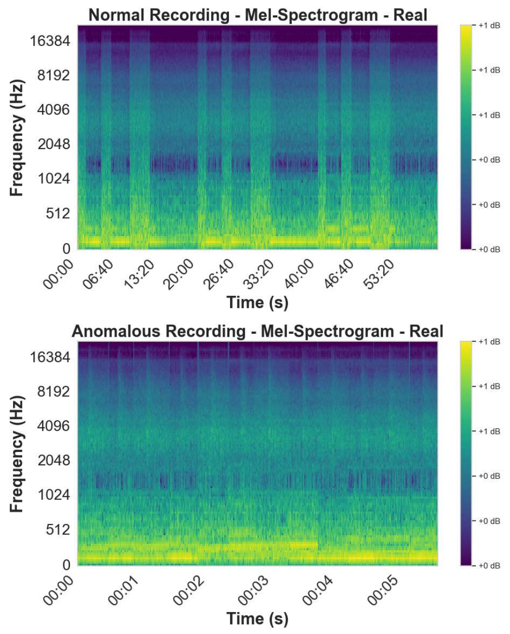


Figure 7: Mel-spectrograms for real dataset: normal (top) vs. anomalous (bottom).

Evaluation metrics. In the real-world scenario, the confusion matrices (cf. Figure 8) confirm that all models successfully detected the transient event during the transition phase, with LSTM-AE and OC-SVM achieving perfect recall. LSTM-AE correctly identified all 43 transient events samples with 150 false positives, achieving the highest F1-score (0.360). OC-SVM also detected all anomalies but misclassified 243 normal samples (F1-score: 0.261). K-Means missed 6 anomalies with 156 false positives, achieving the lowest F1-score (0.314). Per Table 3, K-Means remains fastest for training and inference, while OC-SVM is significantly slower at inference.

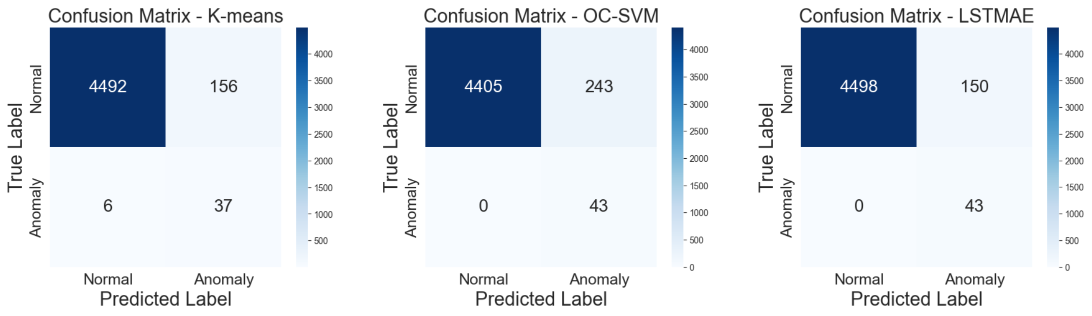


Figure 8: Confusion matrices: Real Industrial Dataset

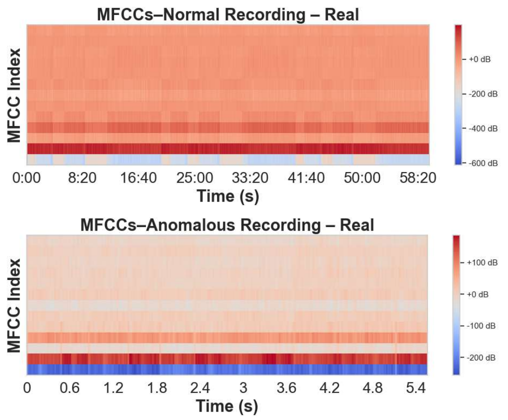


Figure 9: Comparison of Real Recordings MFCCs

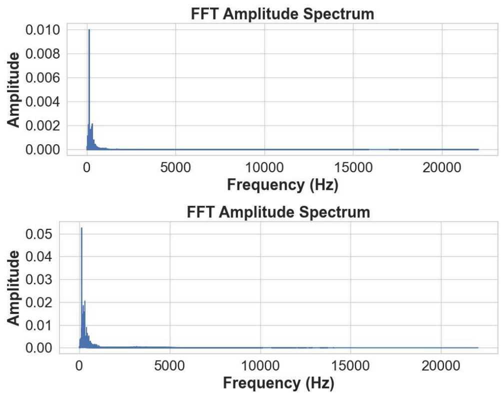


Figure 10: Comparison of Real Recordings FFT Amplitude

of over 0.050dB. This is due to the longer audio file for the normal recording and more distributed acoustic signals in contrast to the short and noisy signals in the anomalous recording.

Table 3: Evaluation metrics for the real world dataset.

Method	Train Time (s)	ROC AUC	Precision	Recall	F1-Score	Inference Time (s)
K-Means	5.25	0.962	0.192	0.861	0.314	0.043
OC-SVM	55.54	0.998	0.150	1.000	0.261	12.17
LSTM-AE	145.25	0.997	0.219	1.000	0.360	0.867

4.3 Comparative Analysis

The experimental evaluation outlines the crucial role of effective audio feature extraction. Mel-spectograms and MFCCs provided key inputs by distinguishing subtle frequency variations, aligning with our exploratory data analysis. This preprocessing stage directly influenced model performance in detecting anomalies despite typical hydropower challenges like background noise and signal reflections.

Model selection involves notable **trade-offs** among accuracy, computational efficiency, and explainability:

- **K-means** offered fast inference (0.027–0.043s) for real-time applications but limited robustness to overlapping acoustic patterns (precision: 0.192).
- **OC-SVM** demonstrated strong detection (ROC AUC: 0.9976 – 0.998) with moderate computational overhead (training time: 2.8–55.5s) and intuitive boundary-based interpretability.
- **LSTM Autoencoder (AE)** captured complex temporal dependencies effectively (ROC AUC: 0.997–0.999) but required significantly higher computational complexity (training time: 145s). Its performance justifies deployment where accuracy is paramount, though explainability and maintenance may pose practical challenges.

5 Conclusions and Future Work

This paper presented an acoustic anomaly detection pipeline for predictive maintenance in PSHP power plants. We benchmark three ML models (K-Means, OC-SVM, and LSTM-AE) on real-world acoustic datasets from Rodundwerk II, analyzing audio feature selection and model trade-offs between accuracy, computational cost, and interpretability. OC-SVM achieved optimal efficiency-accuracy balance, while LSTM-AE excelled at detecting subtle temporal anomalies despite higher computational demands.

Future work includes developing a multi-modal anomaly detection framework that integrates acoustic signals with vibration data and operational parameters (*e.g.*, rotational speed, bearing temperature, power output). Additionally, collecting expanded datasets from multiple microphone arrays positioned across different floors of the plant will enable extending detection capabilities from 2D to 3D, incorporating localization of anomalies.

Acknowledgments

This work was conducted in the context of a joint project with illwerke vkw AG, Bregenz, Austria, to whom the authors express gratitude for all support and discussions on technical aspects, and access to the Rodundwerk II hydropower plant.

References

[1] Hyojung Ahn and Inchoon Yeo. 2021. Deep-Learning-Based Approach to Anomaly Detection Techniques for Large Acoustic Data in Machine Operation. *Sensors* 21, 5446 (2021). doi:10.3390/s21165446

[2] Barış Bayram, Taha Berkay Duman, and Gökhan Ince. 2021. Real time detection of acoustic anomalies in industrial processes using sequential autoencoders. *Expert Systems* 38, 1 (2021), e12564. doi:10.1111/exsy.12564

[3] M. M. Breunig, H. P. Kriegel, R. T. Ng, and J. Sander. 2000. LOF: Identifying density-based local outliers. *ACM SIGMOD Record* 29, 2 (2000), 93–104. doi:10.1145/335191.335388

[4] Gabriel Coelho, Luís Miguel Matos, Pedro José Pereira, André Ferreira, André Pilastri, and Paulo Cortez. 2022. Deep Autoencoders for Acoustic Anomaly Detection: Experiments with Working Machine and In-Vehicle Audio. *RepositóriUM* (2022). https://repositorium.sdum.uminho.pt/bitstream/1822/81433/1/aeaad2.pdf Accessed: 2025-02-16.

[5] Leon Cohen. 1995. *Time-frequency analysis*. Vol. 778. Prentice Hall PTR New Jersey.

[6] S. Davis and P. Mermelstein. 1980. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28, 4 (1980), 357–366. doi:10.1109/TASSP.1980.1163420

[7] E. Di Fiore, A. Ferraro, A. Galli, V. Moscato, and G. Sperlì. 2022. An anomalous sound detection methodology for predictive maintenance. *Expert Systems with Applications* 209 (2022), 118324. doi:10.1016/j.eswa.2022.118324

[8] T. B. Duman, B. Bayram, and G. İnce. 2020. Acoustic Anomaly Detection Using Convolutional Autoencoders in Industrial Processes. In *14th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2019)*, F. Martínez Álvarez, A. Troncoso Lora, J. A. Sáez Muñoz, H. Quintián, and E. Corchado (Eds.). Springer International Publishing, 432–442. doi:10.1007/978-3-030-20055-8_41

[9] E-control. 2025. Aktueller Marktpreis gemäß § 41 Abs. 1 Ökostromgesetz 2012. https://www.e-control.at/industrie/oeko-energie/oekostrommarkt/marktpreise-gem-paragraph-20#:~:text=Dieser%20Marktpreis%20betr%C3%A4gt%20demzufolge%2097,Quartal%202024).

[10] Sarvarbek Erniyazov, Yongmin Kim, M. Jaleel, and Chang Gyoon Lim. 2024. Comprehensive Analysis and Improved Techniques for Anomaly Detection in Time Series Data with Autoencoder Models. *International Journal on Advanced Science, Engineering and Information Technology* 14, 6 (Dec. 2024), 1861–1867. doi:10.18517/ijaseit.14.6.20451

[11] A. Ferraro, A. Galli, V.L. Gatta, V. Moscato, M. Postiglione, G. Sperlì, and F. Moscato. 2023. Unsupervised Anomaly Detection in Predictive Maintenance using Sound Data. *CEUR Workshop Proceedings* 3478 (2023), 449–458.

[12] illwerke vkw. 2025. Rodundwerk II. https://www.illwerkevkw.at/rodundwerk-ii

[13] M. Khanjari, A. Azarfar, M. H. Abardeh, and E. Alibeiki. 2024. Anomalous sound detection for machine condition monitoring using 3D tensor representation of sound and 3D deep convolutional neural network. *Multimedia Tools and Applications* 83, 15 (2024), 44101–44119. doi:10.1007/s11042-023-17043-9

[14] Nand Kishor, RP Saini, and SP Singh. 2007. A review on hydropower plant models and control. *Renewable and Sustainable Energy Reviews* 11, 5 (2007), 776–796.

[15] F. König, C. Sous, A. Ouald Chaib, and G. Jacobs. 2021. Machine learning-based anomaly detection and classification of acoustic emission events for wear monitoring in sliding bearing systems. *Tribology International* 155 (2021), 106811. doi:10.1016/j.triboint.2020.106811

[16] Claudia Linnhoff-Popien, Steffen Illium, Fabian Ritz, and Robert Müller. 2021. Acoustic Anomaly Detection for Machine Sounds based on Image Transfer Learning. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (Volume 2)*, Ana Paula Rocha, Luc Steels, and Jaap van den Herik (Eds.). SciTePress, Setúbal, Portuagl, 49–56.

[17] F. Lo Scudo, E. Ritacco, L. Caroprese, and G. Manco. 2023. Audio-based anomaly detection on edge devices via self-supervision and spectral analysis. *Journal of Intelligent Information Systems* 61, 3 (2023), 765–793. doi:10.1007/s10844-023-00792-2

[18] Slaney Malcolm. 1998. Auditory toolbox version 2. *https://engineering. purdue. edu/~ malcolm/interval/1998-010/* (1998).

[19] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. 2015. librosa: Audio and music signal analysis in python.. In *SciPy*. 18–24.

[20] M. Meire and P. Karsmakers. 2019. Comparison of Deep Autoencoder Architectures for Real-time Acoustic Based Anomaly Detection in Assets. In *2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*, Vol. 2. 786–790. doi:10.1109/IDAACS.2019.8924301

[21] R. Keith Mobley. 2002. 1 - Impact of Maintenance. In *An Introduction to Predictive Maintenance (Second Edition)* (second edition ed.), R. Keith Mobley (Ed.). Butterworth-Heinemann, Burlington, 1–22. doi:10.1016/B978-075067531-4/50001-4

[22] Juan I Pérez-Díaz, Manuel Chazarra, Javier García-González, Giovanna Cavazzini, and Anna Stoppato. 2015. Trends and challenges in the operation of pumped-storage hydropower plants. *Renewable and Sustainable Energy Reviews* 44 (2015), 767–784.

[23] M. Portnoff. 1980. Time-frequency representation of digital signals and systems based on short-time Fourier analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28, 1 (1980), 55–69. doi:10.1109/TASSP.1980.1163359

[24] Harsh Purohit, Ryo Tanabe, Takashi Ichige, Tomoya Endo, Yasunori Nikaido, Kohei Suefusa, and Yasue Kawaguchi. 2019. MIMII dataset: Sound dataset for malfunctioning industrial machine investigation and inspection. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE2019)*. doi:10.33682/m76f-d618

[25] V N Ganapathi Raju, K Prasanna Lakshmi, Vinod Mahesh Jain, Archana Kalidindi, and V Padma. 2020. Study the Influence of Normalization/Transformation process on the Accuracy of Supervised Classification. In *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*. 729–735. doi:10.1109/ICSSIT48917.2020.9214160

[26] ScienceDirect. 2025. Audio Segmentation. https://www.sciencedirect.com/topics/engineering/audio-segmentation Accessed: 2025-02-13.

[27] Siemens. 2023. The True Cost of Downtime: Identify, Avoid, and Overcome. https://assets.new.siemens.com/siemens/assets/api/uuid:3d606495-dbe0-43e4-80b1-d04e27ada920/dics-b10153-00-7600truecostofdowntime2022-144.pdf Digital Industries, Customer Services.

[28] Y. Tagawa, R. Maskeliūnas, and R. Damaševičius. 2021. Acoustic anomaly detection of mechanical failures in noisy real-life factory environments. *Electronics* 10, 19 (2021). doi:10.3390/electronics10192329

[29] Han Wang, Dongdong Wang, Haoxiang Liu, and Gang Tang. 2022. A predictive sliding local outlier correction method with adaptive state change rate determining for bearing remaining useful life estimation. *Reliability Engineering and System Safety* 225 (2022), 108601. doi:10.1016/j.ress.2022.108601

[30] Yiming Yang. 1999. An Evaluation of Statistical Approaches to Text Categorization. *Information Retrieval* 1, 1-2 (1999), 69–90. doi:10.1023/A:1009982220290

[31] Haoru Zhao, Baoshan Zhu, and Boshuang Jiang. 2025. Comprehensive assessment and analysis of cavitation scale effects on energy conversion and stability in pumped hydro energy storage units. *Energy Conversion and Management* 325 (2025), 119370.

[32] Tiago Zonta, Cristiano André da Costa, Rodrigo da Rosa Righi, Miromar José de Lima, Eduardo Silveira da Trindade, and Guann Pyng Li. 2020. Predictive maintenance in the Industry 4.0: A systematic literature review. *Computers and Industrial Engineering* 150 (2020), 106889. doi:10.1016/j.cie.2020.106889

From Latency to Engagement: Technical Synergies and Ethical Questions in IoT-Enabled Gaming

Kurt Horvath
{kurt}.{horvath}@aau.at
University of Klagenfurt
Institute of Information Technology
Austria

Tom Tucek
{tom}.{tucek}@aau.at
University of Klagenfurt
Institute of Information Technology
Austria

ABSTRACT

The convergence of video games with the Internet of Things (IoT), artificial intelligence (AI), and emerging 6G networks creates unprecedented opportunities and pressing challenges. On a technical level, IoT-enabled gaming requires ultra-low latency, reliable quality of service (QoS), and seamless multi-device integration supported by edge and cloud intelligence. On a societal level, gamification increasingly extends into education, health, and commerce, where points, badges, and immersive feedback loops can enhance engagement but also risk manipulation, privacy violations, and dependency. This position paper examines these dual dynamics by linking technical enablers, such as 6G connectivity, IoT integration, and edge/AI offloading, with ethical concerns surrounding behavioral influence, data usage, and accessibility. We propose a comparative perspective that highlights where innovation aligns with user needs and where safeguards are necessary. We identify open research challenges by combining technical and ethical analysis and emphasize the importance of regulatory and design frameworks to ensure responsible, inclusive, and sustainable IoT-enabled gaming.

CCS CONCEPTS

• Social and professional topics → Corporate surveillance; **• Networks → Network mobility**; **• Human-centered computing → Ubiquitous and mobile computing theory, concepts and paradigms**; **Mobile computing**.

KEYWORDS

IoT, Edge AI, Gamification, 6G

1 BACKGROUND AND MOTIVATION

Since the early days of personal computing, video games have been a catalyst for technological innovation, continuously driving advances in both hardware and software [24, 29]. In the 1980s and 1990s, gaming interactions were primarily confined to human-computer interaction. This paradigm shifted in the 2000s, as broadband Internet and wireless networking enabled online multiplayer experiences. Gaming was now also defined by social encounters between geographically distributed players. The rise of mobile devices, tablets, and specialized consoles further diversified gaming platforms, while recent innovations, such as 5G-enabled handheld devices (e.g., *Razer Edge 5G*) and standalone VR headsets (e.g., *Meta Quest 3*) signal a transition toward pervasive, network-native play characterized by seamless connectivity and continuous computation. Today, gaming extends beyond specific platforms or locations, as gamification (the application of game-like designs to non-game contexts) is integrated into daily routines and social practices [14].

The Internet of Things (IoT) introduces a new dimension to this evolution. IoT devices range from low-power sensors to wearables enabled by Artificial Intelligence (AI), and their integration into everyday life creates new opportunities and challenges for interactive applications. Both online games and IoT systems depend on ultra-low latency, scalable computation, and adaptive resource allocation, often requiring a balance between resource-constrained devices and nearby edge infrastructure [16]. AI further reinforces these connections by enabling personalization, real-time analytics, and intelligent orchestration across distributed devices. Beyond technical considerations, the convergence of gaming, IoT, and AI raises ethical and societal questions. Gamification (e.g., collecting points, badges, and leaderboards) has been proven effective in various domains, including education [2], health [23], and commerce [6]. However, while gamification can encourage constructive behaviors, it also carries risks of manipulation, addictive engagement, and the commercialization of user attention [21, 32]. These concerns become particularly pressing in IoT-enabled pervasive computing environments, where gamified interactions are embedded into everyday practices and are difficult to regulate or supervise.

This position paper examines the intersection of these domains by exploring the synergies between gaming, IoT, AI, and 6G. Gaming can serve as a driver for IoT innovation while simultaneously demanding a critical reflection on the ethical implications of pervasive play. Our discussion highlights that regulation should focus on the commercialization of gamification, rather than technical options or gamification itself.

2 CONVERGENCE OF GAMING, IOT, AND AI

The area of ubiquitous computing is established by a large set of connected devices. It is shaped by the demand for interactive experiences with strict latency guarantees [9], the rise of data-driven intelligence, as well as services that incorporate aspects of gaming, IoT, and AI. This section explores the intersection of these technologies and paradigms, where their requirements overlap, and how they jointly shape new services.

2.1 Multi-Device and Pervasive Gaming Ecosystems

Modern gaming is no longer confined to a single device, but spans across multiple devices and platforms [5]. Players interact across a heterogeneous ecosystem of devices, including smartphones, tablets, wearable sensors, AR/VR headsets, and cloud-enabled consoles [31]. This multi-device environment influences IoT system architectures, where devices with varying computational capabilities and connectivity options cooperate to provide a service [17]. Gaming scenarios such as location-based experiences [7] or sensor-driven interaction illustrate how IoT and gaming naturally converge.

2.2 AI-Powered Synergies

AI plays a central role in all three domains, as it enables non-player character behavior, adaptive difficulty, and personalized content delivery in gaming [26, 30]. In IoT, AI supports context recognition, resource optimization, and connectivity [19]. The intersection of these capabilities opens up new opportunities, for example, games that adapt dynamically to player context using IoT sensor data, or IoT services that utilize gaming-inspired interfaces to enhance user engagement. Edge and cloud AI further support these synergies by offloading computationally demanding inference tasks [11].

2.3 Latency, QoS, and QoE Challenges

Both gaming [18] and IoT [13] are highly dependent on low latency and reliable network performance. Latency above a few tens of milliseconds can degrade competitive gaming experiences [4], just as it can reduce the reliability of safety-critical IoT applications. Quality of Service (QoS) guarantees are essential, but ultimately, user-perceived Quality of Experience (QoE) determines success. Predictive adaptation, traffic prioritization, and resource-aware computation offloading are crucial for aligning QoS with QoE in these highly dynamic environments [25, 27].

2.4 The 6G Perspective

Theoretically, 5G mobile networks have already enabled cloud gaming, mobile VR, and latency-aware IoT applications [12]. However, the next generation, 6G, promises to push these boundaries further with sub-millisecond latency, massive device connectivity, integrated edge intelligence, and support for holographic communication [28]. These advances directly benefit IoT and gaming, allowing seamless multi-device integration [33], ultra-realistic immersive environments [8], and adaptive service delivery in highly mobile contexts [10]. Therefore, the envisioned 6G capabilities can act as an enabler of pervasive, AI-enhanced gaming experiences.

3 CHALLENGES AND OPPORTUNITIES

The convergence of gaming, IoT, AI, and 6G causes a reciprocal effect, as many IoT applications become gamified, and games become deeply embedded in everyday life. It presents unprecedented opportunities, but it also raises significant challenges. We analyze these from two complementary perspectives: the *gaming side*, which focuses on technical and experiential requirements, and the *societal side*, which addresses ethical and social issues that can arise from gamifying everyday interactions.

3.1 Gaming-Centric Challenges and Opportunities

From a gaming perspective, the demand for immersive and seamless experiences is the main requirement, for which the following technical conditions must be fulfilled:

- **Latency:** Most games demand real-time responsiveness. Latency above 25 ms can disrupt gameplay and diminish Quality of Experience (QoE) [4]. IoT and 6G offer opportunities for ultra-low-latency, distributed architectures that can support next-generation gaming.
- **Device heterogeneity:** Players increasingly use smartphones, VR headsets, consoles, and IoT-enabled wearables [3]. The challenge lies in ensuring cross-device interoperability, while the opportunity lies in leveraging IoT for pervasive, multi-device play.
- **Edge/AI integration:** Offloading computation to the edge or using on-device AI enables more dynamic and adaptive gameplay. However, this raises challenges in resource allocation, synchronization, and power consumption.
- **Scalability:** Multiplayer and persistent online worlds generate massive traffic. 6G and IoT can provide scalable communication backbones, although challenges remain in ensuring fairness and QoS. This aspect also demands for edge computing concepts.

3.2 Societal and Ethical Challenges of Gamification

Beyond technical concerns, gamification increasingly shapes education, commerce, and daily life. This pervasiveness introduces dilemmas such as:

- **Behavioral influence:** Gamification mechanisms such as points, badges, and rankings can foster engagement, but also risk manipulation and over-dependence, particularly in children and vulnerable users [15].
- **Commercialization of play:** While games can create positive experiences, their integration into IoT systems may blur boundaries between entertainment and commercial exploitation.
- **Privacy:** IoT-enabled gamification relies heavily on personal data to adapt experiences accordingly. This raises issues of trust, transparency, and informed consent [20].
- **Equity and accessibility:** If gaming becomes a pervasive mode of interaction, access to devices and services becomes a factor of social inclusion or exclusion (e.g., paywalls [22])

3.3 A Comparative View: Challenges, Opportunities, and Technical Enablers

We provide an overview highlighting the interplay between technical innovation and social responsibility (Figure 1). This figure summarizes the challenges and opportunities from gaming and societal perspectives, with the technical enablers identified in Section 2 and challenges we identified in Section 3. The aim is to illustrate where technology aligns with user needs and where additional safeguards and design considerations are required.

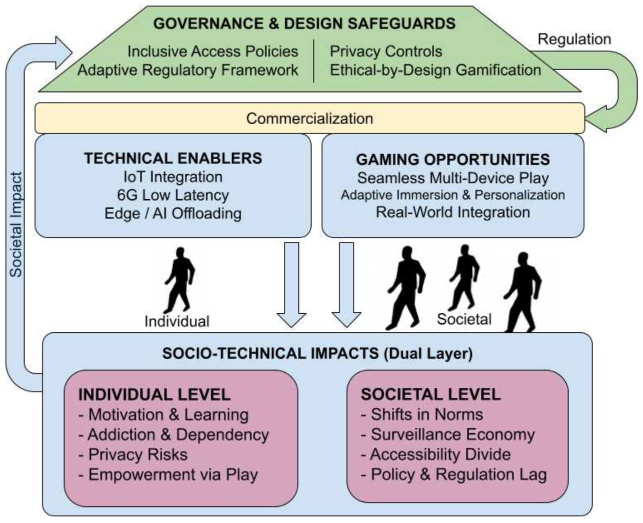


Figure 1: Integration of technological and ethical challenges in society and regulation.

From a technical point of view, the problems of latency, scalability, and device heterogeneity appear to be solvable. Emerging technologies such as 6G and edge computing already demonstrate the ability to reduce latency, increase throughput, and manage large-scale device integration [13]. With continued innovation, the gaming ecosystem can overcome most infrastructure-related constraints and deliver seamless IoT-enabled experiences. Integrating these enablers with gaming methods, multi-device support (including IoT), and immersion with real-world objects could enable more social gaming experiences. The societal dimension presents far more persistent challenges. Addressing behavioral influence issues, privacy protection, and commercializing play requires more than technology, addressing issues of behavioral influence [15], privacy protection [20], and commercialization of technology. To address these concerns, society must first establish a broad awareness of the risks and opportunities such innovations bring to individuals and society [1]. Awareness must extend beyond individual users to enable society and legislative bodies to determine how deeply gaming technologies should impact everyday life. Communicating both the issues and the opportunities, such as the potential for gamification in education and health, ensures that debates and discourse remain aware of the risks.

Awareness, however, is only the first step. Governance must also construct regulatory frameworks that can adapt quickly to technological advancements. We argue that the focus of such regulations should be on the commercialization of technology or gaming concepts, rather than those concepts themselves.

3.4 Future Work

As described in Table 1, future technical research should go beyond improving latency, QoE, and distributed computation to explore the applications that new technical advancements will enable. Societal research must incorporate social impact assessments into the development process, focusing on privacy, autonomy, and fairness. Regulators require clear, evidence-based recommendations to ensure emerging applications align with democratic values, public well-being, and the moral codes of their respective societies. These recommendations should be provided before technologies are widely available and their social implications become apparent.

Table 1: Societal and ethical challenges in IoT-enabled gamification and future research directions.

Technical Research Directions	Societal Research Directions
Focus on privacy-preserving and explainable gamification systems, with adaptive consent mechanisms and safeguards against manipulative design patterns. Detect and mitigate harmful engagement strategies.	Investigate the psychological effects of gamification on various demographics, establish ethical design principles to prevent dependence, and study how users can be supported to balance engagement with well-being.
Develop inclusive and accessible IoT-enabled gaming frameworks, fairness-aware resource allocation models, and transparent monetization strategies that minimize exclusion and promote fairness.	Analyze the cultural implications of the commercialization of play, support the creation of adaptive regulatory frameworks, and explore governance models that ensure access to IoT-enabled experiences.

Subsequently, we should not confine recommendations such as these to academic discourse; we must communicate them to policymakers, industry stakeholders, and the broader public by integrating social and ethical considerations as a core part of technological innovation rather than treating them as separate afterthoughts.

4 CONCLUSION

This position paper has shown that gaming has consistently pushed the boundaries of computing and networking technologies. Its convergence with IoT, AI, and emerging infrastructures defines new opportunities. Advances such as 6G provide credible pathways to resolve many issues, as these enablers can deliver ultra-low latency, reliable QoS, and high QoE, ensuring that the technical foundation for pervasive gaming continues to become stronger. However, societal and technological challenges require deliberate answers. We cannot solve the issues of behavioral influence, privacy, play commercialization, and accessibility with technology alone. We must foster risk awareness while communicating the benefits of gamification in education and health. With such awareness, societies can make democratic decisions about how deeply gaming technologies should influence daily life. At the same time, policymakers must establish adaptive regulatory frameworks that are responsive to changing conditions.

Therefore, the convergence of gaming and IoT requires more than technical progress. It demands social and political processes that evolve together with and in response to innovation.

REFERENCES

[1] Abdullah Algashami, Laura Vuillier, Amen Alrobai, Keith Phalp, and Raian Ali. 2019. Gamification risks to enterprise teamwork: Taxonomy, management strategies and modalities of application. *Systems* 7, 1 (2019), 9.

[2] Ilaria Caponetto, Jeffrey Earp, Michela Ott, et al. 2014. Gamification and education: A literature review. In *European conference on games based learning*, Vol. 1. 50–57.

[3] Entertainment Software Association. 2024. *Essential Facts About the U.S. Video Game Industry 2024*. Technical Report. Entertainment Software Association. https://www.theesa.com/wp-content/uploads/2024/05/Essential-Facts-2024-FINAL.pdf Accessed on 2025-11-04.

[4] David Halbhuber, Annika Köhler, Markus Schmidbauer, Jannik Wiese, and Niels Henze. 2022. The effects of auditory latency on experienced first-person shooter players. In *Proceedings of Mensch und Computer 2022*. 286–296.

[5] Pouya Hamadanian, Doug Gallatin, Mohammad Alizadeh, and Krishna Chintalapudi. 2023. Ekho: Synchronizing cloud gaming media across multiple endpoints. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 533–549.

[6] Juho Hamari, Jonna Koivisto, and Harri Sarsa. 2014. Does Gamification Work? – A Literature Review of Empirical Studies on Gamification. In *2014 47th Hawaii International Conference on System Sciences*. 3025–3034. https://doi.org/10.1109/HICSS.2014.377

[7] Juho Hamari, Aqdas Malik, Johannes Koski, and Aditya Johri. 2019. Uses and gratifications of pokémon go: why do people play mobile location-based augmented reality games? *International Journal of Human–Computer Interaction* 35, 9 (2019), 804–819.

[8] Ananya Hazarika and Mehdi Rahmati. 2023. Towards an evolved immersive experience: Exploring 5G-and beyond-enabled ultra-low-latency communications for augmented and virtual reality. *Sensors* 23, 7 (2023), 3682.

[9] Kurt Horvath, Dragi Kimovski, and Radu Prodan. 2025. SCAREY: Location-Aware Service Lifecycle Management. In *2025 IEEE International Conference on Edge Computing and Communications (EDGE)*.

[10] Kurt Horvath, Dragi Kimovski, Christoph Uran, Helmut Wöllik, and Radu Prodan. 2023. MESDD: A Distributed Geofence-Based Discovery Method for the Computing Continuum. In *Euro-Par 2023: Parallel Processing*, José Cano, Marios D. Dikaiakos, George A. Papadopoulos, Miquel Pericàs, and Rizos Sakellariou (Eds.). Springer Nature Switzerland, Cham, 125–138.

[11] Kurt Horvath, Shpresa Tuda, Blerta Idrizi, Stojan Kitanov, Fisnik Doko, and Dragi Kimovski. 2025. 6G Infrastructures for Edge AI: An Analytical Perspective. In *2025 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 1066–1072. https://doi.org/10.1109/IPDPSW66978.2025.00167

[12] Kurt Horvath, Shpresa Tuda, Blerta Idrizi, Stojan Kitanov, Fisnik Doko, and Dragi Kimovski. 2025. 6G Infrastructures for Edge AI: An Analytical Perspective. *arXiv preprint arXiv:2506.10570* (2025).

[13] Kurt Klaus Horvath, Dragi Kimovski, Bernd Spiess, Oliver Hohlfeld, and Radu Prodan. 2025. SEAL-CC: Scalable Latency Evaluation Methodology for Internet-of-Things Services. In *Proceedings of the 14th International Conference on the Internet of Things (IoT '24)*. Association for Computing Machinery, New York, NY, USA, 117–126. https://doi.org/10.1145/3703790.3703804

[14] Bohyun Kim. 2015. The popularity of gamification in the mobile and social era. *Library Technology Reports* 51, 2 (2015), 5–9.

[15] Jihoon Kim and Darla M Castelli. 2021. Effects of gamification on behavioral change in education: A meta-analysis. *International journal of environmental research and public health* 18, 7 (2021), 3550.

[16] Dragi Kimovski, Roland Mathá, Josef Hammer, Narges Mehran, Hermann Hellwagner, and Radu Prodan. 2021. Cloud, Fog, or Edge: Where to Compute? *IEEE Internet Computing* 25, 4 (2021), 30–36. https://doi.org/10.1109/MIC.2021.3050613

[17] Asif Ali Laghari, Hang Li, Yin Shoulin, Awais Khan Jumani, Abdullah Ayub Khan, and Fida Hussain Dahri. 2025. Internet of Things for gaming: A review. *Entertainment Computing* 52 (2025), 100910.

[18] Shengmei Liu, Mark Claypool, Atsuo Kuwahara, Jamie Sherman, and James J Scovell. 2021. Lower is better? The effects of local latencies on competitive first-person shooter game players. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–12.

[19] Sicong Liu, Bin Guo, Cheng Fang, Ziqi Wang, Shiyan Luo, Zimu Zhou, and Zhiwen Yu. 2023. Enabling resource-efficient AIoT system with cross-level optimization: A survey. *IEEE Communications Surveys & Tutorials* 26, 1 (2023), 389–427.

[20] Aikaterini-Georgia Mavroeidi, Angeliki Kitsiou, Christos Kalloniatis, and Stefanos Gritzalis. 2019. Gamification vs. privacy: Identifying and analysing the major concerns. *Future Internet* 11, 3 (2019), 67.

[21] Tobias Nyström. 2021. Exploring the darkness of gamification: you want it darker? In *Intelligent computing*. Springer, 491–506.

[22] Michael A Peters. 2020. Digital socialism or knowledge capitalism? , 10 pages. https://doi.org/10.1080/00131857.2019.1593033

[23] Lamyae Sardi, Ali Idri, and José Luis Fernández-Alemán. 2017. A systematic review of gamification in e-Health. *Journal of biomedical informatics* 71 (2017), 31–48.

[24] João Paulo Sousa, Rogério Tavares, João Pedro Gomes, and Vitor Mendonça. 2022. Review and analysis of research on Video Games and Artificial Intelligence: a look back and a step forward. *Procedia Computer Science* 204 (2022), 315–323. https://doi.org/10.1016/j.procs.2022.08.038

[25] Maria Daniela Tache, Ovidiu Păscuțoiu, and Eugen Borcoci. 2024. Optimization algorithms in SDN: Routing, load balancing, and delay optimization. *Applied Sciences* 14, 14 (2024), 5967.

[26] Tom Tucek. 2024. Enhancing empathy through personalized AI-driven experiences and conversations with digital humans in video games. In *Companion Proceedings of the 2024 Annual Symposium on Computer-Human Interaction in Play*. 446–449.

[27] Uchenna Joseph Umoga, Enoch Oluwademilade Sodiya, Ejike David Ugwuanyi, Boma Sonimitiem Jacks, Oluwaseun Augustine Lottu, Obinna Donald Daraojimba, Alexander Obaigbena, et al. 2024. Exploring the potential of AI-driven optimization in enhancing network performance and efficiency. *Magna Scientia Advanced Research and Reviews* 10, 1 (2024), 368–378.

[28] Cheng-Xiang Wang, Xiaohu You, Xiqi Gao, Xiuming Zhu, Zixin Li, Chuan Zhang, Haiming Wang, Yongming Huang, Yunfei Chen, Harald Haas, John S. Thompson, Erik G. Larsson, Marco Di Renzo, Wen Tong, Peiying Zhu, Xuemin Shen, H. Vincent Poor, and Lajos Hanzo. 2023. On the Road to 6G: Visions, Requirements, Key Technologies, and Testbeds. *IEEE Communications Surveys and Tutorials* 25, 2 (2023), 905–974. https://doi.org/10.1109/COMST.2023.3249835

[29] Yufei Wang, Xiaodong Xie, Shuai Chen, et al. 2024. A Comprehensive Analysis of NVIDIA's Technological Innovations, Market Strategies, and Future Prospects. *International Journal of Information Technologies and Systems Approach* 17, 1 (2024), 1–26.

[30] Georgios N Yannakakis and Julian Togelius. 2018. *Artificial intelligence and games*. Vol. 2. Springer.

[31] Mingyue Yu, Anfeng Liu, Neal N Xiong, and Tian Wang. 2020. An intelligent game-based offloading scheme for maximizing benefits of IoT-edge-cloud ecosystems. *IEEE Internet of Things Journal* 9, 8 (2020), 5600–5616.

[32] José P. Zagal, Staffan Björk, and Chris Lewis. 2013. Dark Patterns in the Design of Games. In *Proceedings of the 2013 Foundations of Digital Games (FDG)*. Chania, Greece.

[33] Shangwei Zhang, Jiajia Liu, Hongzhi Guo, Mingping Qi, and Nei Kato. 2020. Envisioning device-to-device communications in 6G. *IEEE Network* 34, 3 (2020), 86–91.

WebAssembly on Resource-Constrained IoT Devices: Performance, Efficiency, and Portability

Mislav Has
mislav.has@fer.hr
Faculty of Electrical Engineering and Computing
Zagreb, Croatia

Tao Xiong
tao.xiong@tietoevry.com
Tietoevry Sweden Service AB
Stockholm, Sweden

Fehmi Ben Abdesslem
fehmi.ben.abdesslem@ri.se
Rise Research Institutes of Sweden AB
Stockholm, Sweden

Mario Kušek
mario.kusek@fer.hr
Faculty of Electrical Engineering and Computing
Zagreb, Croatia

Abstract

The increasing heterogeneity of hardware and software in the Internet of Things (IoT) poses a major challenge for the portability, maintainability and deployment of software on devices with limited resources. WebAssembly (WASM), originally designed for the web, is increasingly recognized as a portable, secure and efficient runtime environment that can overcome these challenges. This paper explores the feasibility of using WASM in embedded IoT systems by evaluating its performance, memory footprint and energy consumption on three representative microcontrollers: the Raspberry Pi Pico, the ESP32 C6 and the nRF5340. Two lightweight WASM runtimes, WAMR and wasm3, are compared with the native C execution. The results show that while the native execution remains superior in terms of speed and energy efficiency, WASM offers acceptable trade-offs in return for cross-platform compatibility and sandbox execution. The results highlight that WASM is a viable option for embedded IoT applications when portability and security outweigh strict performance constraints, and that further runtime optimization could extend its practicality in this area.

Keywords

WebAssembly, Internet of Things, Resource-Constrained Devices, Runtime Performance, Energy Consumption

1 Introduction

The Internet of Things (IoT) ecosystem is characterized by a large and growing diversity of hardware platforms, operating systems and development environments. While this heterogeneity enables a wide range of applications, it also introduces developers with significant challenges when it comes to maintaining and deploying software on resource-constrained devices. Traditional approaches often require platform-specific code or extensive rework to ensure compatibility and efficiency on each target. In this context, WebAssembly (WASM) has emerged as a promising technology for standardising development in the IoT landscape.

Originally designed for the web, WASM is a compact, binary instruction format that enables high-performance execution of code compiled from languages such as C, C++, and Rust. Its main strength lies in its portability: the same WASM binary can be executed on different platforms and architectures with minimal changes, given a compatible runtime environment is available. This opens up the possibility to develop once and deploy anywhere, which is a particularly compelling proposition for the fragmented and resource-constrained world of embedded IoT.

In recent years, the WebAssembly ecosystem has expanded beyond browsers into areas such as cloud computing, edge computing, and now embedded systems. This shift has been supported by the development of lightweight WASM runtimes that can run on microcontrollers and small IoT platforms. As a result, WASM is no longer limited to high-end environments and is being seriously considered as a unifying layer for portable and efficient embedded application development.

While WASM is increasingly promoted for its portability in IoT, there remains a lack of systematic evaluation of its performance and energy implications on resource-constrained microcontrollers. This paper addresses this gap by experimentally assessing the execution time, memory footprint, and energy consumption of representative WASM runtimes compared to native execution across multiple IoT devices. In doing so, it provides a quantitative analysis of WASM performance on low-power embedded platforms, examines the runtime-level differences between lightweight interpreters such as wasm3 and more feature-rich runtimes like WAMR, and discusses the trade-offs between portability, performance, and energy efficiency. The experimental design and methodology are based on prior work conducted in a thesis project in [1].

The paper is structured as follows: Section 2 provides an overview of existing research to establish the foundation and relevance of this study. Section 3 introduces the basic concepts of WebAssembly and explores its applicability to IoT environments. Section 4 describes the setup and methodology of the experimental environment. Section 5 presents and analyzes the experimental results. Finally, Section 6 concludes the paper by summarizing key findings and implications.

2 Related Work

There are several papers and projects investigating IoT devices and WASM. Authors in [2] conducted a controlled experiment by compiling three benchmarking algorithms from four different programming languages (i.e. C, Rust, Go, and JavaScript) compiled to WASM and running these languages with two different WASM runtimes on a Raspberry Pi 3B. They identified C and Rust as a solid option for WASM projects working with low-performance hardware traditionally used in IoT devices.

The paper in [3] has presented the possibility of combining WASM with embedded devices. They discuss the WASM runtimes that embedded devices can support and their structure. Authors in [4] have conducted tests on a Raspberry Pi and show that there are many cases where a WASM runtime outperforms a similar Docker + C solution. To address the challenge of running WebAssembly on resource-constrained IoT devices, the paper [5] introduces a lightweight runtime environment tailored for device-and cloud-integrated applications. Their approach supports Ahead-of-Time (AOT) compilation of WebAssembly on such constrained platforms, employing memory reduction techniques and compile-time safety checks to ensure secure sandboxed execution. The results show that this approach reduces memory footprint by up to 84.8x and improves energy efficiency by 1.2x to 4.9x compared to existing AOT runtimes. Moreover, authors in [6] conducted a controlled experiment in which they tested WASM binaries generated from C, Rust, Go, and JavaScript on a Raspberry Pi 3B using two different runtimes. Their results show that the choice of source language has a significant impact on both energy consumption and execution performance, while the runtime environment plays a less decisive role. In particular, C and Rust performed better than other languages, while JavaScript compiled with Javy showed poor efficiency.

Further studies, such as [7], have explored hardware-assisted WebAssembly execution for embedded systems, demonstrating that dedicated acceleration units can substantially improve throughput and efficiency on low-power processors. In addition, [8] provided an extensive review of WebAssembly beyond the web, examining its performance characteristics, runtime constraints, and optimization opportunities in edge and resource-constrained environments.

While these studies have advanced the understanding of WebAssembly performance on single-board computers and moderately constrained platforms, few have focused specifically on ultra-constrained microcontrollers typical of IoT end devices. This paper differs from prior work in three main ways:

(1) It extends the experimental scope to multiple low-power microcontrollers (Raspberry Pi Pico, ESP32 C6, and nRF5340) to capture performance variations across architectures;

(2) It directly compares lightweight and feature-rich WASM runtimes (wasm3 and WAMR) under identical workloads to identify trade-offs in execution time, energy consumption, and memory usage;

(3) It documents practical challenges encountered when deploying and executing WASM on these devices, providing implementation insights useful for future research and development.

By addressing these aspects, this study complements existing research and contributes new empirical data and methodological guidance for evaluating WASM in resource-constrained IoT environments.

3 WebAssembly Background and IoT Integration

WebAssembly is a binary instruction format for a stack-based virtual machine. It was developed as a portable target for compiling high-level languages such as C, C++, and Rust, so that these languages can be used on the web for both client and server applications. While JavaScript remains the only programming language natively supported by web browsers, WASM fills this gap by providing a solution for running safe, fast, and portable low-level code on the web. Therefore, it works alongside JavaScript and often serves as a complementary technology. to improve performance and enable the execution of more complex tasks. The basic design goals of WASM are as follows [9]:

- Fast transfers over the Internet: Considering JavaScript is a plain text format, it cannot achieve the compact file sizes that WASM can. WASM is designed for compact and load-time saving binary representation, which reduces load time and saves bandwidth.
- Security: Security is crucial for web technologies because the code comes from untrusted sources. WASM has a memory-safe execution environment (sandbox) that can be isolated from the host runtime. Other web technologies such as Javascript use a managed language runtime, which can have a negative impact on performance due to the overhead of ensuring security. Essentially, WASM achieves a balance between security and performance by minimizing the performance degradation normally associated with secure execution.
- Portability: The structure of the WASM binary format is designed to run efficiently on different operating systems and instruction set architectures. When a program is compiled into WASM bytecode, it can be distributed and executed in different environments as long as there is a WASM runtime environment is available.
- Open and debuggable: There are several tools and API support for WASM. WASM can also be printed out in a textual format for debugging, testing, experimenting, optimizing, learning, teaching and writing programs by hand. The format allows developers to view the source code of WASM modules on the web, making it more accessible for various development tasks.
- Part of the open web platform: Another design goal of WASM is to maintain the versionless and feature-tested nature of the web. Its modules can be called into and out of the JavaScript context and access browser functionality.

While the Internet of Things is driving technological progress in many areas, it still faces some major challenges. One major issue is scalability and diversity. IoT systems consist of numerous devices and infrastructures, each using different architectures, protocols, and data formats. This leads to major difficulties in ensuring interoperability and seamless integration in large-scale, heterogeneous environments. Security and data protection are also pressing concerns. IoT networks are often the target of attacks, and although there are various security measures in place, these are often accompanied by performance degradation. Energy efficiency is another critical challenge. Many IoT devices are battery-powered, and factors such as unnecessary data transfers or constant wireless usage can quickly consume energy, making power optimization essential.

Finally, limited hardware resources are also a constraint. Tiny IoT devices often lack the processing power and memory to run full operating systems. While lightweight real-time operating systems can be used, they do not offer the same level of flexibility or isolation.

WASM offers some features and benefits that are notably impactful for some of the mentioned IoT challenges:

- Flexibility: WASM was developed as a platform-independent bytecode that increases flexibility in IoT development. It allows the same code to run on different devices and platforms, eliminating the need for platform-specific customization. This feature can improve the efficiency of new technology development, which is critical for rapidly evolving IoT ecosystems [9].
- Security and data protection: The usual, widely used memory protection methods are not available for low-end microcontrollers. However, by implementing WASM on these devices, the problem can be solved without the need for virtual memory and extensive hardware resources. The reason for this is that WASM provides an abstraction layer that allows applications to run in a sandbox environment so that the code can be executed in a controlled environment. This can help protect the host system from potentially malicious operations [3].
- Compactness and Energy Efficiency: WASM has a binary instruction set, which can be directly processed by the device, saving energy for resource-intensive operations. This efficiency feature can lead to lower energy consumption for CPU usage, especially for battery-powered IoT devices.
- Ahead of Time compilation: Some WASM runtimes support Ahead of Time compilation, which takes the WASM bytecode and produces machine code for the target CPU/MCU type. This is very useful in the context of tiny IoT devices, which may have limited CPU and memory to perform Just in Time compilation, as we typically do in the cloud or on the desktop [10].

There are a variety of WASM runtimes for web and non-web [11], which have features [12], such as performance optimization, portability, and so on. In this study, we focus on runtimes that support IoT devices. Table 1 shows the comparison of the five most popular WASM runtimes. Each of these runtimes has different features and capabilities tailored to different development needs.

Table 1: The most popular WASM runtimes.

Wasm Runtime	Standalone Interpreter	JIT	AOT	GitHub Stars
wasmer[13]	✓	X	✓	19.9k
wasmtime[14]	✓	X	✓	16.6k
WasmEdge[15]	✓	X	X	9.6k
wasm3[16]	✓	✓	X	7.6k
WAMR[17]	✓	✓	✓	5.4k

Due to its robust support for Just-in-Time (JIT) and Ahead-of-Time (AOT) compilation, WAMR is the first choice to improve performance and flexibility. JIT compilation converts the WASM code into machine code at runtime to improve performance. AOT compilation compiles WASM code into native machine code before execution, reducing startup times. In addition, these runtimes have a large community, resulting in better support, frequent updates, and a more extensive ecosystem, which is crucial for academic and practical implementations. WAMR has more comprehensive support for all the features, which makes it versatile for comprehensive applications. Additionally, it supports various IoT platforms, which makes them ideal for research on lightweight, cross-platform runtime environments in resource-constrained devices. Compared with other runtimes, wasmedge and wasm3 offer a better balance between integration capabilities and runtime efficiency. Specifically, wasm3 focuses more on speed with selective compilation features, while wasmedge pays more attention to interoperability.

4 Implementation

To evaluate the execution of WASM on resource-constrained devices, we conducted experiments with two runtimes: WAMR and wasm3. All measurements were performed with interpretation enabled, not ahead-of-time (AOT) or just-in-time (JIT) compilation, to better reflect the capabilities and constraints typical of embedded environments. Selection of the runtime for each platform was based on its availability, maturity, and compatibility. WAMR was chosen for the Raspberry Pi Pico, ESP32 C6, and nRF5340 due to its flexibility and Zephyr integration, while wasm3 was used for the Pico and ESP32 C6 due to its lightweight and fast interpretation capabilities. To ensure a meaningful comparison between different environments and languages, we tested various computational algorithms implemented in both C and Rust. These included a bubble sort function to represent control-flow intensive workloads and a CRC-16 checksum calculation to simulate bitwise and performance-critical operations. The choice of these workloads reflects typical patterns in embedded applications, illustrating the trade-offs developers face between control-flow complexity and computational intensity. These examples allowed us to assess runtime behaviour, performance, and compatibility under different code structures and computational workloads. The following subsections describe the setup and execution of the experiments for each platform.

4.1 Experiments Execution on Pico

4.1.1 Wasm3 experiments. The entire process of running wasm3 on Pico is shown in Figure 1. At the beginning, there is a C or Rust code in the project, which is used as source code for the generation of WASM code. Next, a bash script is executed. This script compiles the C or Rust code into WASM code with the help of the Clang compiler from the WASI SDK. At the same time, it generates a WAT file (WebAssembly Text Format) to enable better analysis of the WASM code. Subsequently, xxd is used to convert a binary file (WASM file) into a C header file format. The xxd is then used to convert the binary WASM file into a C header format. A separate C file as an interface to connect the WASM code to the C code embedded in Pico. The generated C header file is integrated into this interface code, and the final output is a .uf2 file that can be flashed directly to the Pico and can be transferred.

4.1.2 WAMR experiments. The entire process of running WAMR on Pico is shown in Figure 2. First, C or Rust code is compiled into WASM code. Then, a bash script is executed to generate WAT code and a C header file. Finally, West, a tool from Zephyr, is used to

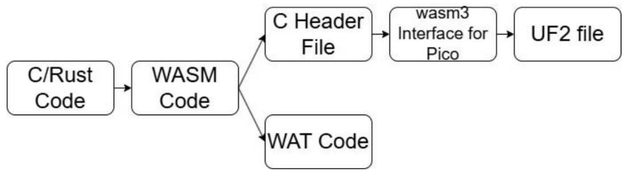


Figure 1: Wasm3 execution on Pico.

build and flash the project in Pico. For the C and Rust projects, the same examples are selected that are used for Pico with wasm3.

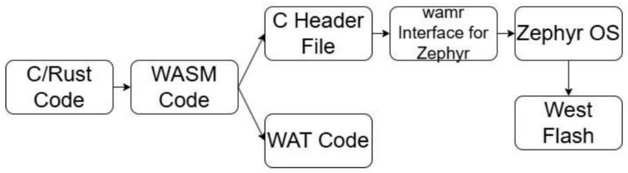


Figure 2: WAMR execution on Pico.

4.2 Experiments Execution on ESP32 C6

4.2.1 Wasm3 experiments: Running wasm3 with ESP32 C6 follows a process similar to running wasm3 with Pico. However, there is direct support from the wasm3 community for ESP32 C6, which allows running wasm3 on the ESP32 C6 without additional platform support. Consequently, we can utilize the wasm3 ESP32 C6 interface directly, rather than developing a custom interface as was necessary for the Pico.

4.2.2 WAMR experiments. As mentioned previously, after adding support for ESP32 C6 to WAMR, it is now possible to run WASM code directly on ESP32 C6. Consequently, the process is similar to WAMR and Pico, but without using Zephyr. First, C or Rust code is compiled into the corresponding WASM code. Next, it will generate WAT file for analysis and a C header file. Moreover, the header file will then be integrated into the WAMR interface file for ESP32 C6. Finally, ESP tools, primarily Python-based, are used to build the project, flash the code, and monitor the output.

4.3 Experiments Execution on nRF5340

4.3.1 WAMR experiments. Because WAMR does not have direct support for the nRF5340 board, but Zephyr does, the process follows the same steps as the WAMR setup on the Pico, as shown in Figure 2.

5 Performance and energy consumption analysis

To assess the runtime efficiency of WASM on constrained embedded systems, we performed a detailed evaluation of the performance and energy consumption of three microcontrollers: the Raspberry Pi Pico, ESP32 C6, and nRF5340. In these experiments, WAMR and wasm3 runtimes were directly compared to native C execution, which serves as a baseline across all devices.

We tested the performance and energy behavior using code written in both C and Rust. To challenge the runtimes, we implemented several computational workloads: a control-intensive algorithm using bubble sort with a varying number of integers, 100 and 1000, and a CRC-16 algorithm with 100 integers as a data set. Although these benchmarks are simple, they reflect common classes of embedded workloads and provide reproducible test cases that can be easily replicated by others. The results shown in the following figures reflect the consistent pattern observed across all tests: native implementations were the most efficient in terms of energy and execution time, while WASM runtimes, especially wasm3, offered competitive performance at a higher but manageable resource cost.

5.1 Energy consumption measurement

The energy consumption of a device can be determined by multiplying the average power by the duration of the interval during which this average power is calculated. This relationship is expressed by the formula: $E = P_{\text{average}} \times T_{\text{execution}}$. In all experiments, we put the device to sleep for 5 seconds before and after the execution of the benchmark algorithms. In this case, there is a short increase in energy consumption caused by the benchmark algorithm. We repeat the same steps several times to obtain reliable results. With this approach, we can calculate the energy consumption by determining the average power consumption during the active phase and multiplying it by the execution time of the function. The product of these values gives the total energy consumption for the execution of the function. All measurements were repeated several times to minimize the effect of variability and to confirm consistency across runs. The formula is used for all three tiny IoT devices in the project to easily compare the results. Therefore, we measure the average current and voltage supplied, as the power can be calculated using the formula P = V × I. Then we multiply these values by the execution time to determine the energy consumption. In this context, the Power Profiler Kit II [18], also known as PPK2, is used to measure the power consumption of all three devices. PPK2 is a stand-alone device capable of measuring and optionally delivering currents in the range of sub-microamperes up to 1 ampere. It can be used with all Nordic Development Kits (DKs) as well as with external hardware. With the help of the PPK2, we can monitor the current in real time and determine the peak current value. The PPK2 operates at a high sampling rate, which ensures that short-term peaks in current consumption are accurately detected. The following graph in Figure 3 shows the energy consumption of the Bubble Sort algorithm (100 integers, implemented in C) on three microcontrollers.

The native C implementation consistently has the lowest energy consumption across all platforms. The values range from 0.06 mJ on the Pico to 0.1 mJ on the ESP32 C6 and the nRF5340. In contrast, WebAssembly runtimes have a significantly higher energy consumption. On the Raspberry Pi Pico, wasm3 consumed 3.11 mJ. On the ESP32 C6, wasm3 consumed 1.12 mJ, while WAMR consumed 2.96 mJ. On the nRF5340, only WAMR was supported and the energy consumption was measured at 2.82 mJ. The WAMR runtime shows a relatively constant energy consumption on both the ESP32 C6 and the nRF5340 (2.96 mJ and 2.82 mJ respectively), indicating stable behavior across different architectures. However,

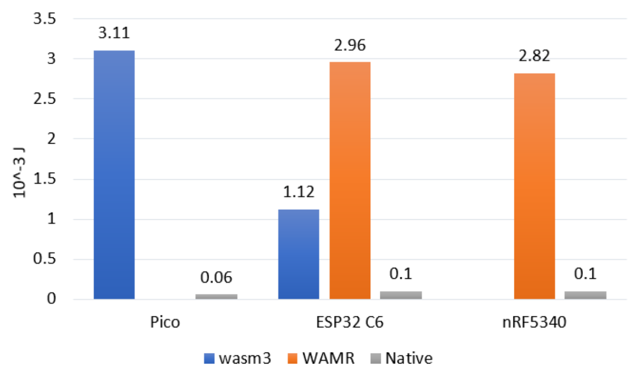


Figure 3: Energy consumption of devices with Bubble Sort algorithm.

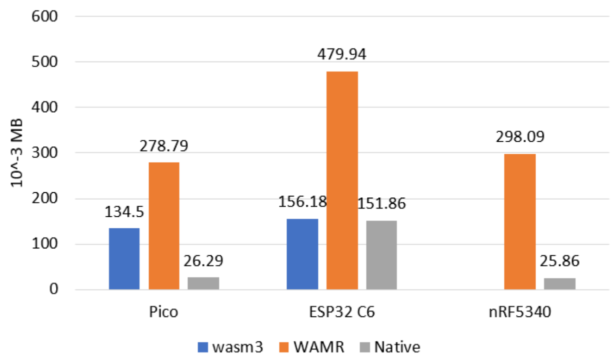


Figure 4: Memory footprint of devices with Bubble Sort algorithm.

wasm3 exhibits a significant difference between the platforms. Energy consumption drops significantly from the Pico (3.11 mJ) to the ESP32 C6 (1.12 mJ). This difference can be attributed to the underlying microarchitecture or the hardware power management features. This emphasises an important trade-off: WASM runtimes offer cross-platform compatibility, but at the cost of higher energy consumption on devices with limited resources.

5.2 Memory Footprint Measurement

The methods of memory footprint measurement are different for the various devices, as they use different methods for flashing the firmware.

5.2.1 Pico: The static memory footprint for the Pico consists of the size of the UF2 file and the RAM usage during program execution. The size of the UF2 file indicates the memory allocated in the flash memory of the Pico, while the RAM usage refers to the memory required by the program while running. First, the source code is compiled to create a UF2 binary file for flashing the firmware to the Pico. The size of the UF2 file, which represents the static memory in the microcontroller's flash memory, is recorded. During program execution, RAM usage is monitored using profiling tools or development environment utilities. The maximum RAM usage is recorded in order to determine the maximum dynamic memory requirement.

5.2.2 ESP32 C6: The IDF toolchains include the 'idf.py size' command, which provides a comprehensive summary of memory usage that includes the dimensions of text and data sections. Therefore, this command can be used directly to quantify the memory requirements of the project.

5.2.3 nRF5340: Upon completion of the project build, Zephyr displays a detailed summary of the project's memory usage. For the purposes of this study, which requires an assessment of static memory usage, the cumulative total of FLASH and RAM regions utilized is considered. This comprehensive measurement includes all allocated memory segments and provides a clear and precise evaluation of the memory resources consumed by the project.

The memory footprint comparison across platforms for the 100-integer bubble sort task shows consistent trends in runtime memory requirements. The native implementations occupy the smallest amount of memory on all three devices, confirming their minimal overhead. Among the WebAssembly runtimes, WAMR consistently shows the highest memory consumption, with values of about 279 KB on the Pico, almost 480 KB on the ESP32 C6, and 298 KB on the nRF5340. This high consumption reflects the additional memory required to manage the WASM environment and the runtime abstraction layers. In contrast, wasm3 has a more compact memory profile, especially on the Pico (about 134 KB) and the ESP32 C6 (156 KB), and is not available on the nRF5340 in this test. The relatively smaller gap between wasm3 and native code suggests that it may be a more memory-efficient choice for constrained systems, although it is still significantly heavier than native execution.

5.3 Execution Time Measurement

The execution time measured refers to the time required by the benchmark algorithms for their execution. For WASM code, it is defined as the interval between the start of the execution of the function by the WASM runtime environment and the termination of the function. For native code, the execution time is measured from the time the function is executed until it is completed. In addition, different IoT devices and WASM runtimes have different libraries. Therefore, each device uses its own method to measure the execution time. The comparison of execution times for the 100-integer bubble sort algorithm on the Pico, ESP32 C6, and nRF5340 platforms is visible in Figure 5.

The native code achieves the fastest execution times on all devices: 611.75 µs (Pico), 577.5 µs (ESP32 C6), and 864.06 µs (nRF5340), demonstrating the advantage of running directly on the hardware with minimal overhead. WASM runtimes, while naturally introducing some overhead due to their abstraction layer, demonstrate encouraging performance given their runtime features and flexibility. Wasm3 consistently delivers solid execution times, for example, 6,358 µs on the ESP32 C6 compared to 577.5 µs natively, making it a practical choice when balancing performance and runtime functionality. While WAMR has higher execution times (e.g., 29,475 µs on Pico), it offers a rich feature set and supports a wider range of

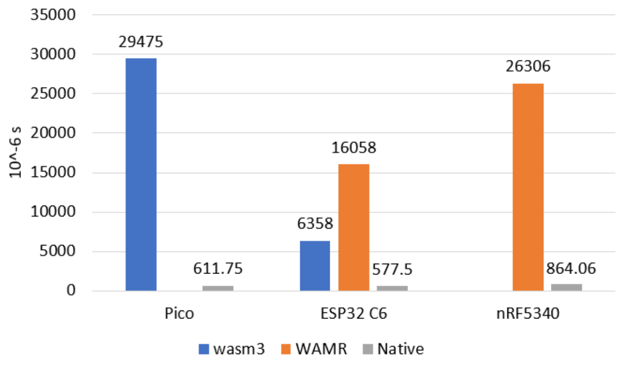


Figure 5: Execution time of devices with Bubble Sort algorithm.

development scenarios, making it a good choice for more complex or modular applications.

5.4 Discussion

Although native execution consistently outperforms WASM runtimes in raw speed and energy efficiency, the results highlight several scenarios in which WASM offers distinct advantages for IoT systems. These advantages lie mainly in portability, safety, and maintainability across heterogeneous devices rather than pure performance.

In heterogeneous IoT deployments, devices often differ in architecture and operating systems, complicating firmware updates and maintenance. WASM mitigates this by providing a portable, architecture-independent execution layer that allows the same bytecode to run on multiple platforms with minimal modification. This feature simplifies over-the-air (OTA) updates and enables uniform code deployment across diverse hardware. Moreover, WASM's sandboxed execution model isolates untrusted code, improving security for devices that must execute third-party or dynamically loaded modules, such as industrial gateways or shared sensor nodes. Its modularity also allows IoT gateways to dynamically load or update functionalities without reflashing firmware, supporting adaptive and long-lived systems.

The comparative analysis of runtimes shows that wasm3 and WAMR serve complementary purposes. wasm3 is a lightweight interpreter optimized for constrained environments, offering fast startup, low memory usage, and predictable behavior. These qualities make it ideal for simple, battery-powered end devices or control tasks where deterministic execution and minimal footprint are critical [11]. In contrast, WAMR provides a richer feature set, AOT and JIT compilation, multi-threading, and integration with operating systems like Zephyr [19]. These capabilities make WAMR better suited for edge or gateway devices that can leverage runtime extensibility and partial compilation to improve performance despite higher overhead. However, this study focused exclusively on the interpretation mode for both runtimes, which reflects the operational reality of highly resource-constrained IoT hardware. The omission of JIT and AOT execution modes represents a limitation, as these approaches can substantially improve performance and reduce energy consumption in more capable devices.

Finally, wasm3 prioritizes minimalism and efficiency for resource-limited devices, while WAMR emphasizes flexibility and integration for more capable embedded systems. Although both runtimes trail native execution in raw performance, their security, portability, and maintainability make them strong candidates for scalable, cross-platform IoT development.

6 Conclusion

The experimental results show a consistent trend: native implementations outperform WASM on key performance metrics, including execution time, energy consumption and often memory usage. However, performance varied significantly between WASM runtimes. Lighter-weight runtimes, such as wasm3, generally executed algorithms faster and with a lower memory footprint than heavier runtimes such as WAMR. These differences illustrate how runtime design and optimization significantly affect efficiency, especially on resource-constrained hardware. While certain platforms struggled to run WASM due to limited memory capacities or slower processing capabilities, others proved to be more powerful and better supported. Devices with larger memory capacity and broader runtime compatibility were generally better suited for WASM deployment, especially for more complex or memory-intensive applications.

Despite its current limitations in performance and energy efficiency, WASM remains a compelling option in scenarios where portability, security and cross-platform development are priorities. For applications where resource conservation or low-latency execution is critical, native code remains the preferred approach. However, as WASM runtimes continue to mature and embedded hardware evolves, the trade-offs become more favorable. This study concludes that while WASM is not yet a complete replacement for native execution in constrained environments, it is an increasingly practical solution, especially when development flexibility and platform independence are desired. These findings can also guide developers in selecting the most suitable WASM runtime for their specific hardware and application requirements, balancing efficiency, memory usage, and feature support.

Future work will explore a broader range of WASM runtimes to better assess their suitability for diverse IoT use cases, especially as the technology continues to mature. This includes evaluating additional classes of resource-constrained devices to gain deeper insight into the practical challenges of deploying and executing WASM in such environments. Future studies should also conduct a systematic comparison of interpreted, AOT, and JIT execution modes to quantify their respective trade-offs in execution speed, memory footprint, and power efficiency under varying IoT workloads. Another promising direction is the integration of real-time operating systems (RTOS) with WASM, which could enable more responsive and capable embedded applications. As both WASM runtimes and embedded hardware continue to evolve, these efforts will help clarify the role of WASM in the next generation of IoT systems.

7 Acknowledgments

This work has been supported by the Horizon Europe WIDERA program under the grant agreement No. 101079214 (AIoTwin).

References

[1] Tao Xiong. Performance Measurement of WebAssembly on IoT Constrained Devices. Master's thesis, KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science (EECS), Stockholm, Sweden, 2024.

[2] Linus Wagner, Maximilian Mayer, Andrea Marino, Alireza Soldani Nezhad, Hugo Zwaan, and Ivano Malavolta. On the Energy Consumption and Performance of WebAssembly Binaries across Programming Languages and Runtimes in IoT. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, EASE '23, page 72–82, New York, NY, USA, 2023. Association for Computing Machinery.

[3] Stefan Wallentowitz, Bastian Kersting, and Dan Mihai Dumitriu. Potential of WebAssembly for Embedded Systems. In *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–4, 2022.

[4] Fredrik Eriksson and Sebastian Grunditz. Containerizing WebAssembly: Considering WebAssembly Containers on IoT Devices as Edge Solution, 2021. [Page 19.].

[5] Borui Li, Hongchang Fan, Yi Gao, and Wei Dong. Bringing Webassembly to resource-constrained IoT devices for seamless device-cloud integration. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, MobiSys '22, page 261–272, New York, NY, USA, 2022. Association for Computing Machinery.

[6] Linus Wagner, Maximilian Mayer, Andrea Marino, Alireza Soldani Nezhad, Hugo Zwaan, and Ivano Malavolta. On the energy consumption and performance of Webassembly binaries across programming languages and runtimes in IoT. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, pages 72–82, 2023.

[7] Jinyeol Kim, Raehyeong Kim, Jongwon Oh, and Seung Eun Lee. Hardware-Based WebAssembly Accelerator for Embedded System. *Electronics*, 13(20), 2024.

[8] Sangeeta Kakati and Mats Brorsson. WebAssembly Beyond the Web: A Review for the Edge-Cloud Continuum. In *2023 3rd International Conference on Intelligent Technologies (CONIT)*, pages 1–8, 2023.

[9] Partha Pratim Ray. An Overview of WebAssembly for IoT: Background, Tools, State-of-the-Art, Challenges, and Future Directions. *Future Internet*, 15(8), 2023.

[10] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with WebAssembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, page 185–200, New York, NY, USA, 2017. Association for Computing Machinery.

[11] Yixuan Zhang, Mugeng Liu, Haoyu Wang, Yun Ma, Gang Huang, and Xuanzhe Liu. Research on WebAssembly Runtimes: A Survey. *ACM Trans. Softw. Eng. Methodol.*, January 2025.

[12] Appcypher. Awesome Wasm Runtimes: A Curated List of Awesome WebAssembly Runtimes, Frameworks, and Tools. https://github.com/appcypher/awesome-wasm-runtimes, 2024. Accessed: 2025-07-06.

[13] Wasmer. Wasmer: The Fastest WebAssembly Runtime. https://github.com/wasmerio/wasmer, 2024. Accessed: 2025-07-06.

[14] Bytecode Alliance. Wasmtime: A Standalone Wasm-Only Optimizing Runtime, Designed to Efficiently Run WebAssembly on a Variety of Platforms from Desktops to IoT Devices. https://github.com/bytecodealliance/wasmtime, 2024. Accessed: 2025-07-06. [Page 12.].

[15] WasmEdge. WasmEdge: A Lightweight, High-Performance, and Extensible WebAssembly Runtime. https://github.com/WasmEdge/WasmEdge, 2024. Accessed: 2025-07-06. [Page 12.].

[16] Wasm3. Wasm3: A High Performance WebAssembly Interpreter. https://github.com/wasm3/wasm3, 2024. Accessed: 2025-07-06. [Page 12.].

[17] Bytecode Alliance. Wasm Micro Runtime (WAMR): A Lightweight WebAssembly Runtime Optimized for Embedded Devices. https://github.com/bytecodealliance/wasm-micro-runtime, 2024. Accessed: 2025-07-06. [Page 12.].

[18] Nordic Semiconductor. Power Profiler Kit II. https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2, 2024. Accessed: 2025-07-08. [Page 38.].

[19] WAMR documentation. WAMR Project Documentation. https://wamr.gitbook.io/document/basics/introduction/wamr_project. (accessed 2025-30-10).

Serverless Everywhere: A Comparative Analysis of WebAssembly Workflows Across Browser, Edge, and Cloud

Mario Colosi
MIFT Department, University of Messina
Messina, Italy

Reza Farahani
Institute of Information Technology, University of Klagenfurt
Klagenfurt, Austria

Lauri Lovén
Center for Ubiquitous Computing, University of Oulu
Oulu, Finland

Radu Prodan
Department of Computer Science, University of Innsbruck
Innsbruck, Austria

Massimo Villari
MIFT Department, University of Messina
Messina, Italy

Abstract

WebAssembly (Wasm) is a binary instruction format that enables portable, sandboxed, and near-native execution across heterogeneous platforms, making it well-suited for serverless workflow execution on browsers, edge nodes, and cloud servers. However, its performance and stability depend heavily on factors such as startup overhead, runtime execution model (e.g., Ahead-of-Time (AOT) and Just-in-Time (JIT) compilation), and resource variability across deployment contexts. This paper evaluates a Wasm-based serverless workflow executed consistently from the browser to edge and cloud instances. The setup uses wasm32-wasi modules: in the browser, execution occurs within a web worker, while on Edge and Cloud, an HTTP shim streams frames to the Wasm runtime. We measure cold- and warm-start latency, per-step delays, workflow makespan, throughput, and CPU/memory utilization to capture the end-to-end behavior across environments. Results show that AOT compilation and instance warming substantially reduce startup latency. For workflows with small payloads, the browser achieves competitive performance owing to fully in-memory data exchanges. In contrast, as payloads grow, the workflow transitions into a compute- and memory-intensive phase where AOT execution on edge and cloud nodes distinctly surpasses browser performance.

CCS Concepts

• **Computing methodologies → Distributed algorithms**.

Keywords

Serverless Computing, Workflow, WebAssembly, Edge Computing, Browser-Edge-Cloud Continuum.

1 Introduction

In the serverless paradigm, providers manage the scalability, placement, and lifecycle of short-lived, event-driven functions. While this abstraction significantly reduces the operational overhead for developers, it exposes drawbacks, including cold starts, queue latency, performance variability, and cost unpredictability [6]. Optimizing these aspects requires consistent measurement methodologies across the computing continuum, i.e., browsers, edge instances, and cloud servers [7]. Recent works have analyzed the causes of cold starts, offering a deeper understanding of their cost implications and underscoring the need for rigorous comparative evaluations [9].

WebAssembly (Wasm), with its compact bytecode, fast validation and instantiation, and strong sandboxed isolation, serves as a unifying execution substrate across the computing continuum [5, 20]. Recent advancements, such as the maturation of runtimes and the standardization of the *WebAssembly System Interface* (WASI) [1], have brought the "*compile once, run anywhere*" paradigm closer to reality. In practice, the same Wasm artifact (i.e., a compiled and portable binary module) can be executed across diverse contexts without modification, offering two key benefits: *(i)* methodological comparability and *(ii)* a simplified software supply chain with uniform packaging and minimal system dependencies.

Recent studies confirm the competitiveness of Wasm compared to solutions like containers and Function-as-a-Service (FaaS) platforms, particularly under resource-constrained conditions [3]. However, trade-offs remain in terms of startup latency and memory footprint, indicating that Wasm's advantages are not uniform across all deployment contexts. Comparisons between x86 and ARM architectures further demonstrate the growing maturity of this technology while revealing runtime variations with practical implications for system design and development choices [11]. Nevertheless, existing evaluations are restricted to a single environment or microbenchmark, offering limited insight into the comparison of end-to-end latency (i.e., function cold/warm behavior, function execution and communication time, and workflow makespan) for realistic application workflows.

To our knowledge, no unified methodological work has yet executed the same Wasm-based workflow across browsers, edge nodes, and cloud platforms using consistent and homogeneous metrics. Thus, this paper presents a comparative analysis of serverless workflows implemented in Wasm and executed across three distinct environments: the browser, the edge, and the cloud. The entire evaluation is performed under a unified experimental setup, ensuring methodological consistency and enabling a fair, cross-environment comparison of performance and resource efficiency. The results of this analysis provide practical guidance for making informed orchestration decisions (e.g., function placement) in serverless workflows. The key contributions of this paper are:

[1]https://wasi.dev

(i) we implement a serverless workflow in Wasm, exposing WASI interfaces compatible with three environments, browser, edge, and cloud;
(ii) we isolate the effects of the execution environment and quantify workflow performance on cold and warm starts, function makespan and workflow makespan, throughput, and resource utilization (CPU, memory);
(iii) we compare the performance of ahead-of-time (AOT) and just-in-time (JIT) as compilation strategies, along with a pre-warming method, to evaluate their impact within each environment.

This paper has six sections: Section 2 summarizes the background and related work on Wasm, serverless computing, workflow processing, and cold start latency reduction. Section 3 describes the strategy adopted for browsers, edge instances and cloud servers. Section 4 explains our evaluation setup before describing the experimental results in Section 5. Section 6 finally concludes the paper.

2 Related Work

This section reviews related work in three main areas: Wasm as an execution substrate for serverless platforms, serverless workflow orchestration strategies, and cold-start mitigation and cost modeling in serverless systems.

2.1 Wasm as a serverless execution substrate

Recent measurements show that Wasm can serve as a competitive alternative to, or an integration layer with, containers [17]. However, its maturity and compatibility still vary across runtime implementations and their integration with the underlying execution environment [12]. Systematic comparison of Wasm runtimes ranks standalone and browser-integrated executors across execution models, interpreter, just-in-time (JIT), and ahead-of-time (AOT) compilation, as well as system interfaces and security aspects, highlighting significant differences in initialization latency and throughput [21]. These findings guide the selection of runtimes and configuration options, such as AOT and JIT compilation modes, as well as caching strategies evaluated in our experiments.

On the edge side, benchmarks show that Wasm-based serverless platforms can achieve reduced startup times and competitive efficiency profiles compared to widely used container-based solutions [14]. However, methodological differences in workloads, metrics, and network configurations across studies highlight the need for a homogeneous measurement protocol [3]. In parallel, recent analyses show the maturation of client-side, in-browser execution [2, 15]. While Wasm offers advantages in portability and time-to-use for scientific computing, it remains constrained by limited tooling and restricted system access [16]. Furthermore, in-browser large language model (LLM) inference systems demonstrate that compute-intensive operations can be executed directly on the client using WebGPU for acceleration and Wasm for CPU-bound computation, achieving near-native performance on the same hardware [1, 19].

2.2 Serverless workflow orchestration strategies

On the other hand, existing research on serverless workflow orchestration investigates the effects of placement strategies on workflow makespan and resource consumption [7, 8, 18]. Edge resource allocation mechanisms using serverless platforms typically employ formalized policies derived from observable runtime metrics [13], making placement decisions repeatable and reproducible by third parties under identical inputs and configurations. The comparative work of centralized and distributed orchestration approaches has demonstrated that the underlying architecture significantly influences workflow completion times and performance variability [4]. A recurring observation across these works is the necessity of empirical characterization of behaviors, such as cold and warm start patterns, queuing latency, and resource footprint—supported by homogeneous benchmarking. Therefore, without such standardized measurements, transferring placement strategies across heterogeneous environments becomes unreliable. These insights motivate our focus on comparable measurements of the same workflow executed in the browser, at the edge, and in the cloud.

2.3 Cold-start and cost management

One of the key factors in serverless computing is the *cold start*, referring to the additional latency that occurs when a function invoked without an already warm instance available [6]. This forces the platform to provision a new sandbox environment (e.g., microVM or container), load and initialize the runtime along with its dependencies, and bring the function code to a ready-to-execute state. For Wasm, this process includes module validation, compilation, and instantiation. However, recent work [9] shows that although cold start can sometimes dominate end-to-end latency in certain production workloads, many functions are not latency-sensitive. Consequently, the authors argue that serverless systems should be evaluated and designed with a broader perspective, considering not only startup latency but also predictability, throughput, and resource efficiency. With respect to cost, recent work [10] investigates how pricing models, billing granularity, and data egress fees affect the total cost of serverless applications. These works introduce parameterized evaluation methodologies that facilitate normalized comparisons across providers, yet require workload and runtime measurements for proper calibration.

2.4 Contributions beyond the state-of-the-art

Prior works evaluated Wasm runtimes, serverless orchestration, and cold-start behaviors largely in isolation, focusing on a single environment (browser, edge, or cloud) or on micro-benchmarks. To our knowledge, no unified, cross-environment work has executed the same Wasm-based workflow across these three layers under a common experimental setup and set of metrics. This paper instead provides an end-to-end comparison of Wasm-based serverless workflows executed in the browser, at the edge, and in the cloud, offering new insights into how runtime configurations (AOT, JIT) and environmental factors jointly affect performance and resource efficiency across the computing continuum.

3 Experimental Design

This section explains how we evaluate the impact of the execution environment (browser, edge, and cloud instances), on the performance of Wasm-based serverless workflows under a consistent measurement setup.

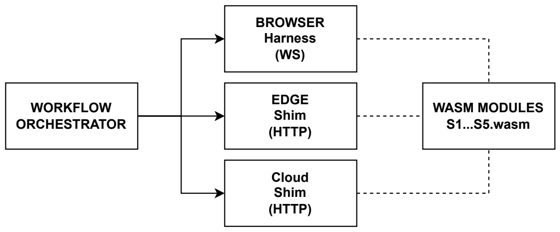


Figure 1: High-level execution pipeline of the serverless workflow across browser, edge, and cloud environments.

Workflow artifacts: We model the application as a directed acyclic graph (DAG), where each node represents a Wasm module with explicit input/output interfaces. The same artifacts are executed across all environments, allowing us to attribute observed performance differences solely to environmental factors rather than to code or packaging variations.

Workflow orchestrator: We developed an external component that serves as a workflow orchestrator, interpreting the DAG and triggering each task according to its structure (e.g., sequence, fan-out, and fan-in). Each task's output is passed as input to its successors, ensuring that tasks remain isolated and do not invoke one another directly. This design preserves identical flow control across all three environments. The orchestrator also records timing and resource utilization metrics, enabling consistent and comparable measurements across experiments. The orchestrator runs in a control environment isolated from the workflow Wasm executors to prevent interference. In the browser setup, it is co-located on the same machine but runs in a separate process, communicating via a loopback WebSocket with a harness page that forwards messages to the Web Worker. The Web Worker executes the workflow tasks in a separate thread. At the edge, the orchestrator runs on a different machine within the same LAN as the executor node and invokes tasks through a lightweight HTTP shim co-located with the Wasm runtime. In the cloud configuration, it runs on a virtual machine (VM) in the same region, using the same HTTP shim interface to trigger executions. Fig 1 illustrates the resulting high-level software architecture.

Metrics and data collection We organize the collected metrics into two complementary perspectives for evaluating application performance. The first captures end-to-end workflow behavior and results, while the second focuses on resource utilization and runtime-level effects within the platform:

(i) User-level metrics:

- *Cold start*: invocation without an existing warm instance, requiring sandbox provisioning (e.g., container), artifact loading, runtime initialization, and dependency setup;
- *Warm start*: served by an already initialized instance without reloading or reinitialization;
- *Function Latency*: Start/end timestamps per step;
- *Workflow makespan*: total elapsed time from acquiring the first step's input to producing the final output, including internal queuing and inter-step data transfers.

(ii) System-level metrics:

- *Throughput*: number of completed workflow invocations per unit time under steady-state conditions with fixed concurrency;
- *Overhead and resources*: CPU (average and peak utilization), memory (RSS and peak), I/O activity, and artifact size (Wasm binary and AOT cache), with the sampling period and measurement tools specified.

Analysis and reproducibility: We adopt a robust statistical analysis, reporting median, 25th/75th percentiles (p25/p75), interquartile range (IQR), and, where applicable, 95 % confidence intervals for key performance indicators. To ensure fair comparisons, we explicitly specify the varying factors between measurements (e.g., execution environment or ablation parameters) while keeping all other parameters constant. In addition, reproducibility is guaranteed through strict artifact versioning, fixed runtime configurations, and deterministic random seeds.

4 Evaluation Setup

This section explains how we executed experiments across browsers, edge, and cloud instances under identical conditions, i.e., Wasm artifact, invocation Application Binary Interface (ABI), and workflow orchestrator. To accommodate browser constraints, the workflow is designed to operate without file-system access. The ABI remains identical across all environments; only the transport mechanism differs: WebSocket with `postMessage` in the browser, and HTTP `multipart/form-data` at the edge and in the cloud instances.

Each workflow task is implemented in Rust and compiled into Wasm modules using the same toolchain (Rust $\longrightarrow$ wasm32-wasi), allowing AOT and JIT compilation to be toggled as ablation parameters. All builds and corresponding module images are version-controlled via commit hashes or container digests for reproducibility. Compilation flags and enabled Wasm features are kept identical across the environments for consistent execution behavior.

4.1 Workflow

We evaluate a serverless workflow, modeled as a DAG with ($N = 5$) functions and fan-out/fan-in $\langle f_{\text{out}}, f_{\text{in}} \rangle = \langle 4, 4 \rangle$, as shown in Fig. 2. As mentioned, each step is implemented in Rust and compiled to `wasm32-wasi` modules that exchange data frames encoded in the Concise Binary Object Representation (CBOR) format entirely in memory. The workflow processes deterministic synthetic payloads (generated with a fixed seed) in three input sizes: $s_{\text{small}} = 16\,\text{KiB}$, $s_{\text{medium}} = 1\,\text{MiB}$, $s_{\text{large}} = 4\,\text{MiB}$. The workflow functions are:

(S1) *Ingest&Deserialize (serialization-bound):* decodes the CBOR frame in memory buffer, validates the schema, and calculates a CRC32 of the input for a quick integrity check. Complexity cost: $O(n)$.

(S2) *Preprocess (memory-bound):* applies normalization and a scan over the buffer (sliding window, clamp, type conversion `u8`$\rightarrow$`f32`); the output is a buffer of the same size. Complexity cost: $O(n)$;

(S3) *Map (CPU-bound, fan-out=4):* divides the buffer into four blocks and processes them in parallel with a blocked *dense matrix multiplication.* Complexity cost: $O(d^3)$, with $d = \sqrt{s/4}$;

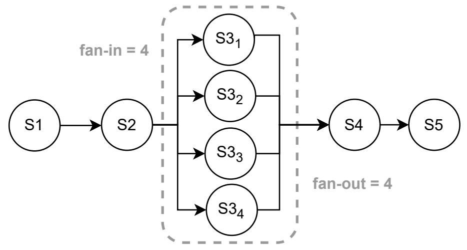


Figure 2: Workflow DAG.

(S4) *Reduce (serialization-bound, fan-in=4):* aggregates the four results (sum/concatenation according to the scheme) into a single block and calculates a partial digest (*BLAKE3*) for verification. Complexity cost: $O(n)$;

(S5) *Serialize&Finalize (serialization-bound):* recodes the final output in CBOR and produces the final digest (*BLAKE3*) to be compared with the expected value. Complexity cost: $O(n)$.

CPU kernel sizing (S3). The size of the matrix multiplication is derived from the payload:

$$d(s) = \begin{cases} 64 & \text{if } s = s_{\text{small}} \ (\approx 64^2 \cdot 4\,\text{B} \simeq 16\,\text{KiB}), \\ 512 & \text{if } s = s_{\text{medium}} \ (\approx 512^2 \cdot 4\,\text{B} \simeq 1\,\text{MiB}), \\ 1024 & \text{if } s = s_{\text{large}} \ (\approx 1024^2 \cdot 4\,\text{B} \simeq 4\,\text{MiB}). \end{cases}$$

Each S3$[k]$ processes a separate block in parallel (fan-out = 4); S4 combines the results.

4.2 Platforms, Runtimes, and Tooling

To ensure consistency across browsers, edges, and clouds, we use the same Wasm artifact (target wasm32-wasi, identical toolchain and flags) and fix runtime versions and configurations. The invocation details are described in Section 4.3; here we summarize the platforms, runtimes, and tools used. Table 1 lists, for each environment, hardware/OS, runtime/browser versions, Wasm flags, and resource limits.

Browser host. The machine running the browser is an Ubuntu 24.04 LTS workstation with an Intel® Core i7-8700K (6 cores, 12 threads, 3.70 GHz) and 16 GiB RAM. We use Mozilla Firefox 143.0.4 (64-bit, Snap for Ubuntu), with WebAssembly SIMD enabled. The workflow is executed in a Web Worker, while the orchestrator runs in a separate process on the same machine and communicates via loopback WebSocket with a harness page.

Edge Node. The edge runs the same module inside a minimal OCI/Docker container with WasmEdge v0.13.5. The host is Ubuntu 24.04 LTS (linux/arm64) with Docker Engine 28.5.1 (cgroups v2); resources are capped via container limits (--cpus 4, --memory 4). Wasm features (SIMD) are aligned with the browser; warm state is achieved through a pool of pre-initialized instances, while cold state is forced by resetting the pool. No disk I/O in the workflow; messages pass through memory to the executor.

Cloud VM. We replicate the edge setup on a linux/amd64 VM co-located with the orchestrator (same region): 4 vCPU, 24 GiB RAM, Ubuntu 24.04; container runtime Docker Engine 28.5.1 (containerd v1.7.28, runc 1.3.0). The Wasm executor is WasmEdge v0.13.5, with the same toolchain configuration as on the edge. Resource limits and cold/warm policies are identical to edge to make the results comparable. Again, the workflow does not access the file system; orchestration uses in-memory messages.

4.3 Execution Environments

We adopt a unified invocation ABI for all environments. Each task receives a request frame using CBOR serialization for metadata fields. On edge/cloud, large binary payloads are transmitted separately via HTTP multipart/form-data to avoid a base64 or similar encoding overhead within the CBOR envelope. This hybrid approach maintains protocol type safety while supporting payloads up to several megabytes without buffer overflow issues. The minimum structure of exchanged messages is as follows:

- request: `{op:"invoke", step_id, run_id, payload}`
- response: `{op:"result", status, payload[, error]}`

The orchestrator sends/waits for these frames and correlates the measurements via `run_id`. Since we do not use the file system in the workflow, input/output travels as buffers in memory.

Browser. The orchestrator communicates with a harness page via WebSocket. The page forwards the frames to the Web Worker with transferable `ArrayBuffer` `postMessage` and sends the Web Worker's response back to the orchestrator. Cold start: new Web Worker instance and cache/Service Worker invalidation; warm start: Web Worker and module already compiled remain alive. Timestamps are collected with `performance.now()`.

Edge. The orchestrator invokes a minimal HTTP endpoint on the edge node: `POST /invoke` (CBOR binary body). A shim in the container delivers the frame to the Wasm executor (WasmEdge) via pipe/IPC and returns the response frame. Cold start: one-shot mode; warm start: pool of pre-initialized instances.

Cloud. Same edge pipeline on VM/container in the same region as the orchestrator: POST /invoke (CBOR binary body, HTTP/1.1) and same cold/warm behavior as above. Timestamps are recorded with a monotonic clock (Rust `Instant`) and the shim returns a CBOR response including the phase breakdown (load, compile, instantiate, init, execute), total latency, and best-effort CPU/RSS samples collected from `/proc`.

4.4 Measurement Protocol

In the experiments, we use a mixed set of micro and macro workloads. The micro workloads include: (i) CPU-bound (e.g., matmul) with synthetic payloads of size $\langle s_{\text{small}}, s_{\text{medium}}, s_{\text{large}} \rangle$; (ii) memory-bound (allocation/scan); (iii) serialization-bound (CBOR $\leftrightarrow$ in–memory). The macro workflow is a DAG with $\langle N \rangle$ steps, fan-out/fan-in $\langle f_{\text{out}}, f_{\text{in}} \rangle$, and explicit I/O scheme; for each step, we specify the format and expected size of input/output.

The measurements of latency, cold/warm behavior, and throughput are performed in a closed-loop configuration. Each configuration (environment $\times$ payload $\times$ mode ablation) is repeated k

Table 1: Platforms and runtimes.

Env.	Host / Instance	CPU	RAM	OS	Browser/Container	Wasm Runtime
Browser	`Workstation (linux/amd64)`	`6 cores`	16 GiB	Ubuntu 24.04	Firefox 143.0.4	`Firefox WebAssembly engine (SpiderMonkey)`
Edge	`Raspberry Pi 4 (linux/arm64)`	`4 cores`	4 GiB	Ubuntu 24.04	Docker 28.5.1 (cgroups v2)	WasmEdge v0.13.5
Cloud	`VM (linux/amd64)`	4 vCPU	24 GiB	Ubuntu 24.04	Docker 28.5.1	WasmEdge v0.13.5

Wasm features: WASI `preview1` (edge/cloud; `JS shim`, subset in browser); SIMD on; Threads on.
Resource limits: Edge/Cloud via cgroups (`--cpus`, `--memory`); Browser via Web Worker isolation.

Table 2: Artifact sizes comparison between WASM modules and their AOT compiled outputs.

Step	Size WASM Module (MB)	Size AOT Artifact (MB)	Size Increase
S1	0.152	0.342	+ 125%
S2	0.135	0.317	+ 135%
S3	0.141	0.322	+ 128%
S4	0.151	0.354	+ 134%
S5	0.153	0.360	+ 135%

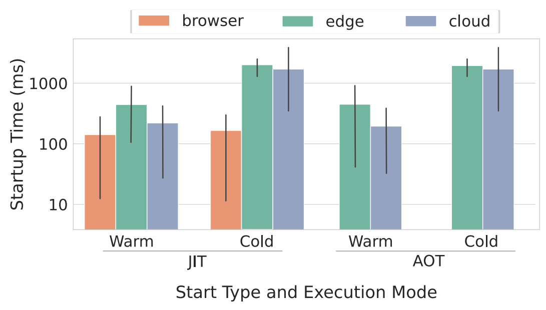


Figure 3: Cold vs Warm startup times.

times ($k = 20$ with randomized order, separate warm-up, and fixed seed. Timestamps are collected with monotonic clocks (browser: `performance.now()`; edge/cloud: high-resolution clock) and correlated via trace-id/span-id; we record: cold-start breakdown (load $\rightarrow$ compile/AOT-load $\rightarrow$ instantiate $\rightarrow$ init $\rightarrow$ run), per-step and end-to-end latencies, workflow makespan, throughput, CPU (average/peak), and memory (RSS/peak).

Outlier, retry, and exclusions follow predefined rules: we discard a run if the monitor detects abnormal conditions; for valid samples, we apply an outlier filter based on $1.5 \times$ IQR.

5 Experimental Results

In this section, we present the experimental results obtained by running the same WebAssembly workflow separately in the browser, edge, and cloud. The analyses are organized along two complementary perspectives: *User-Perceived Performance* and *System-Level Metrics*. The results are aggregated over k repetitions.

Before discussing performance, we quantify the size of the artifacts. Table 2 compares, for each workflow module, the size of the Wasm bytecode and that of the AOT (precompiled) output, also reporting the AOT/Wasm percentage increase; this provides the context for interpreting the benefits observed in cold tests with AOT and for estimating the impact of deployment in the three environments.

5.1 User-Perceived Performance

We analyze the application's performance that directly impacts the user experience, using three predefined metrics: (i) cold/warm startup times, (ii) latencies per workflow step, and (iii) end-to-end makespan.

Cold and warm startup. As shown in Figure 3, warm starts are faster than cold starts in all environments, although this gap is less pronounced in browsers. The use of AOT, which is only applicable on edge/cloud, significantly reduces the amount of time spent on compilation/validation in cold tests. The browser shows particularly low warm start times, benefiting from compilation/instance already residing in the Worker; edge shows the highest cost in cold JIT, while cloud ranks in the middle with slightly lower variability.

Function latency. We run the workflow with a warm profile and 1MiB as payload size and collect the start/end timestamps for each step (S1–S5). In Figure 4 we represent the distributions with boxplots on a logarithmic scale: box = IQR (p25–p75), center line = median; whisker = 1.5×IQR; outliers shown explicitly. The browser performs better on S3–S4–S5 (lower medians and narrower IQRs), thanks to in-memory execution and lower orchestration/IPC overhead. On S1–S2, it ranks between edge and cloud: the overhead of (de)serialization and the Worker-main thread transition (postMessage/buffer transfer) make it less advantageous for the browser, while edge/cloud runtimes with optimized containers and I/O pipes mitigate the cost of these steps. In summary, when computation or aggregation prevails (S3–S4–S5), the browser is competitive; when (de)serialization dominates (S1–S2), the advantage diminishes.

Workflow makespan. We measure the time S1$\longrightarrow$S5 (including queues and transfers) in *warm* mode by varying the *payload size* and the *JIT/AOT* mode. In Figure 5, we observe a clear reversal as the payload increases: with small sizes, the browser (JIT) is often faster because the startup cost is minimal, everything happens in-memory, and Worker orchestration has little impact. Moving to medium/large sizes, the makespan becomes compute/memory-bound: the startup cost is amortized and memory bandwidth, cache, and code quality prevail; here, AOT configurations on edge/cloud are faster and more stable. In particular, edge AOT tends to emerge

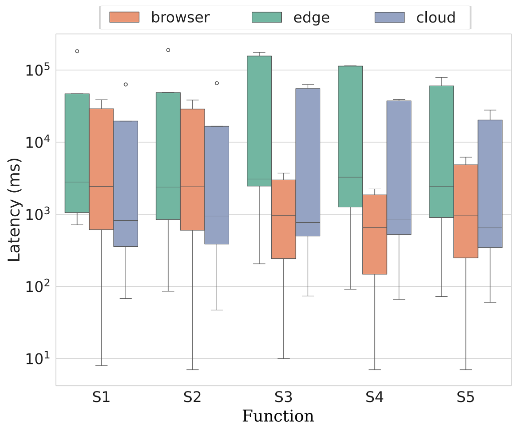


Figure 4: Function latency.

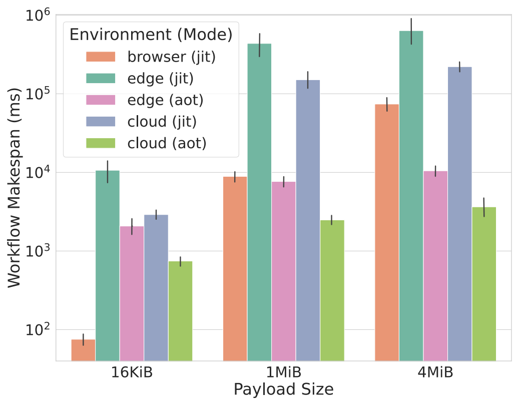


Figure 5: Workflow makespan.

when the load is markedly numerical, while the browser's advantage diminishes until it reverses. In all cases, warm and AOT improve predictability and end-to-end times, with more pronounced effects as size increases.

5.2 System-Level Metrics

This section observes the platform's behavior under load, complementing user-facing metrics. We consider sustainable capacity (throughput) and resource usage profile (CPU/RSS), measured with the same experimental setup.

Throughput. A new invocation is issued upon completion of the previous one, so that the reported value corresponds to the inverse of the average completion time. Figure 6 shows results consistent with those for makespan. With 16 KiB, the browser (JIT) prevails: lightweight invocations, in-memory exchanges, and reduced startup costs allow for tens of requests per second. At 1 MiB and 4 MiB, the compute/memory-bound effect emerges: the initial overhead is amortized, and AOT configurations on edge/cloud outperform the browser due to the elimination of JIT compilation and more favorable CPU/memory-bandwidth/cache.

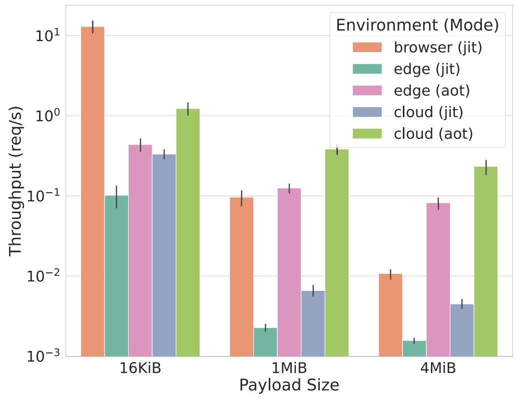


Figure 6: Workflow throughput.

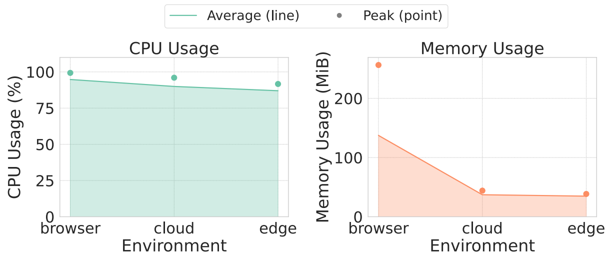


Figure 7: Resource usage.

Resource usage. We sample the CPU (average/peak) and memory (average/peak RSS) of the executor process, with a sampling interval of 20 ms on edge/cloud (readings from /proc) and 20 ms in the browser. In the browser, CPU is obtained by recording a Performance session from in-browser DevTools and aggregating the profile samples; the same tools on the Web Worker side estimate memory. As shown in Figure 7, CPU usage is high in all environments (indicating computational bottlenecks), with average values slightly lower on the edge than in the cloud and browser. Memory usage differs more significantly: the browser has higher average/peak RSS (JS engine stack, Worker, GC management), while the cloud and edge remain more compact thanks to the native Wasm runtime and the absence of browser components.

6 Conclusion

We presented a comparative analysis of the execution of the same serverless workflow in browsers, edge, and cloud, keeping Wasm and ABI artifacts unchanged. The results indicate that warm and AOT significantly reduce startup times and variability. The browser is competitive on small loads and on steps dominated by orchestration/aggregation, while edge/cloud in AOT prevail with increasing payloads when the regime becomes compute/memory-bound. Makespan and throughput consistently reflect these trends, and CPU/RSS profiles explain the differences. This evidence should be interpreted within the scope of the setup (runtimes, hardware, browser, workload) and does not claim to be generalizable beyond the experiment.

Acknowledgment

This work received partial funding from the Italian Ministry of University and Research (MUR) “Research projects of National Interest (PRIN-PNRR)” call through the project “Cloud Continuum aimed at On-Demand Services in Smart Sustainable Environments” (CUP: J53D23015080001- IC: P2022YNBHP), and "SEcurity and RIghts in the CyberSpace (SERICS)" project (PE00000014), under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU (CUP: D43C22003050001), the European Union’s Horizon Europe research and innovation program, grant agreements 101093202 (Graph-Massivizer), and the Austrian Research Promotion Agency (FFG, grant agreement 909989, AIM AT Stiftungsprofessur für Edge AI)

References

[1] Zhiyang Chen, Yun Ma, Haiyang Shen, and Mugeng Liu. 2025. WeInfer: Unleashing the Power of WebGPU on LLM Inference in Web Browsers. In *Proceedings of the ACM on Web Conference 2025* (Sydney NSW, Australia) *(WWW '25)*. Association for Computing Machinery, New York, NY, USA, 4264–4273. doi:10.1145/3696410.3714553

[2] Joao De Macedo, Rui Abreu, Rui Pereira, and João Saraiva. 2021. On the Runtime and Energy Performance of WebAssembly: Is WebAssembly Superior to Javascript Yet?. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 255–262.

[3] Giuseppe De Palma, Saverio Giallorenzo, Jacopo Mauro, Matteo Trentin, and Gianluigi Zavattaro. 2024. WebAssembly at the Edge: Benchmarking a Serverless Platform for Private Edge Cloud Systems. *IEEE Internet Computing* 28, 6 (2024), 37–44. doi:10.1109/MIC.2024.3513035

[4] Abdallah Elshamy, Ahmed Alquraan, and Samer Al-Kiswany. 2023. A Study of Orchestration Approaches for Scientific Workflows in Serverless Computing. In *Proceedings of the 1st Workshop on SErverless Systems, Applications and MEthodologies* (Rome, Italy) *(SESAME '23)*. Association for Computing Machinery, New York, NY, USA, 34–40. doi:10.1145/3592533.3592809

[5] Reza Farahani, Dragi Kimovski, Sashko Ristov, Alexandru Iosup, and Radu Prodan. 2023. Towards Sustainable Serverless Processing of Massive Graphs on the Computing Continuum. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 221–226.

[6] Reza Farahani, Frank Loh, Dumitru Roman, and Radu Prodan. 2024. Serverless Workflow Management on the Computing Continuum: A Mini-Survey. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*. 146–150.

[7] Reza Farahani, Narges Mehran, Sashko Ristov, and Radu Prodan. 2024. Heftless: A Bi-objective Serverless Workflow Batch Orchestration on the Computing Continuum. In *2024 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 286–296.

[8] Reza Farahani and Radu Prodan. 2025. EnergyLess: An Energy-Aware Serverless Workflow Batch Orchestration on the Computing Continuum. In *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*. IEEE, 243–254.

[9] Muhammed Golec, Guneet Kaur Walia, Mohit Kumar, Felix Cuadrado, Sukhpal Singh Gill, and Steve Uhlig. 2024. Cold Start Latency in Serverless Computing: A Systematic Review, Taxonomy, and Future Directions. *Comput. Surveys* 57, 3 (2024), 1–36.

[10] Muhammad Hamza, Muhammad Azeem Akbar, and Rafael Capilla. 2024. Understanding Cost Dynamics of Serverless Computing: An Empirical Study. In *Software Business*, Sami Hyrynsalmi, Jürgen Münch, Kari Smolander, and Jorge Melegati (Eds.). Springer Nature Switzerland, Cham, 456–470.

[11] Sangeeta Kakati and Mats Brorsson. 2024. A Cross-Architecture Evaluation of WebAssembly in the Cloud-Edge Continuum. In *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 337–346. doi:10.1109/CCGrid59990.2024.00046

[12] Mugeng Liu, Haiyang Shen, Yixuan Zhang, Hong Mei, and Yun Ma. 2025. WebAssembly for Container Runtime: Are We There Yet? *ACM Trans. Softw. Eng. Methodol.* 34, 6, Article 174 (July 2025), 22 pages. doi:10.1145/3712197

[13] Samaneh Hajy Mahdizadeh and Saeid Abrishami. 2024. An Assignment Mechanism for Workflow Scheduling in Function as a Service Edge Environment. *Future Generation Computer Systems* 157 (2024), 543–557. doi:10.1016/j.future.2024.04.003

[14] Pankaj Mendki. 2020. Evaluating WebAssembly Enabled Serverless Approach for Edge Computing. In *2020 IEEE Cloud Summit*. IEEE, 161–166.

[15] Biju R Mohan et al. 2022. Comparative Analysis of JavaScript And WebAssembly in the Browser Environment. In *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*. IEEE, 232–237.

[16] Jeffrey M Perkel. 2024. No Installation Required: How WebAssembly Is Changing Scientific Computing. *Nature* 627, 8003 (2024), 455–456.

[17] Steven Pham, Kaue Oliveira, and Chung-Horng Lung. 2023. WebAssembly Modules as Alternative to Docker Containers in IoT Application Development. In *2023 IEEE 3rd International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB)*. IEEE, 519–524.

[18] Sashko Ristov, Reza Farahani, and Radu Prodan. 2023. Large-scale Graph Processing and Simulation with Serverless Workflows in Federated FaaS. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 227–231.

[19] Charlie F Ruan, Yucheng Qin, Xun Zhou, Ruihang Lai, Hongyi Jin, Yixin Dong, Bohan Hou, Meng-Shiun Yu, Yiyan Zhai, Sudeep Agarwal, et al. 2024. WebLLM: A High-Performance In-Browser LLM Inference Engine. *arXiv preprint arXiv:2412.15803* (2024).

[20] Yixuan Zhang, Mugeng Liu, Haoyu Wang, Yun Ma, Gang Huang, and Xuanzhe Liu. 2025. Research on WebAssembly Runtimes: A Survey. *ACM Trans. Softw. Eng. Methodol.* (Jan. 2025). doi:10.1145/3714465

[21] Mircea Ţălu. 2025. A Comparative Study of WebAssembly Runtimes: Performance Metrics, Integration Challenges, Application Domains, and Security Features. *Archives of Advanced Engineering Science* (Apr. 2025), 1–13. doi:10.47852/bonviewAAES52024965

Wi-Fi Enabled Edge Intelligence Framework for Smart City Traffic Monitoring using Low-Power IoT Cameras

Raphael Walcher
{rawalcher}@edu.aau.at
University of Klagenfurt
Institute of Information Technology
Klagenfurt, Austria

Kurt Horvath
{kurt}.{horvath}@aau.at
University of Klagenfurt
Institute of Information Technology
Klagenfurt, Austria

Dragi Kimovski
{dragi}.{kimovski}@aau.at
University of Klagenfurt
Institute of Information Technology
Klagenfurt, Austria

Stojan Kitanov
{stojan}.{kitanov}@unt.edu.mk
Mother Teresa University
Faculty of Information Sciences
Skopje, North Macedonia

ABSTRACT

Real-time traffic monitoring in smart cities demands ultra-low latency processing to support time-critical decisions such as incident detection and congestion management. While cloud-based solutions offer robust computation, their inherent latency limits their applicability for such tasks. This work proposes a localized edge AI framework that connects low-power IoT camera sensors to a client, or applies offloading of inference to an NVIDIA Jetson Nano (GPU). Networking is achieved via Wi-Fi, enabling image classification without relying on wide-area infrastructure such as 5G, or wired networks. We evaluate two processing strategies: local inference on camera nodes and GPU-accelerated offloading to the Jetson Nano. We show that local processing is only feasible for lightweight models and low frame rates, whereas offloading enables near-real-time performance even for more complex models. These results demonstrate the viability of cost-effective, Wi-Fi-based edge AI deployments for latency-critical urban monitoring.

CCS CONCEPTS

• **Networks → Network performance evaluation**; *Network structure*; *Location based services*; • **Computing methodologies →** Distributed computing methodologies.

KEYWORDS

IoT Services, Computing Continuum, Edge AI, Smart City

1 INTRODUCTION

Smart cities function as intelligent systems designed to enhance quality of life by addressing key areas such as mobility and traffic monitoring [1], as well as energy optimization and waste management. Among these applications of smart cities, real-time traffic analysis is central in enabling timely decision-making [2] [19] for incident detection, congestion mitigation, and adaptive signal control. Such applications require ultra-low latency and a large amount of compute resources to be effective [10]. While traditional cloud-based solutions ensure substantial computational power, their inherent network latency and limited Quality of Service (QoS) guarantees pose significant challenges for supporting latency-critical and real-time smart city applications.

Emerging paradigms in edge computing and artificial intelligence (AI) address this challenge by bringing computation closer to the data source [13], thereby reducing latency and bandwidth consumption. However, the deployment of edge systems often assumes the availability of low-latency communication infrastructure such as 5G or fiber networks. These assumptions do not hold universally, particularly in areas with limited coverage or with dense population [10]. Wi-Fi can provide an alternative in densely populated areas [11]. Furthermore, many urban monitoring (and smart city) needs are inherently local, targeting specific intersections, pedestrian zones, or parking facilities. In such cases, wide-area connectivity may be unnecessary or even counterproductive.

This work proposes a localized edge AI framework that uses Wi-Fi as a primary communication technology to connect **low-power IoT camera sensors** with shared edge devices to offload AI inference on vehicle classification.

Each camera node captures high-resolution images and processes them locally using its CPU or offloads them to Nvidia Jetson Nano nodes for GPU-accelerated inference. The design prioritizes nearby, high-performance processing within the coverage range of a Wi-Fi network, eliminating reliance on wide-area infrastructure.

The main contributions of this work are:

- Design and implementation of a Wi-Fi–based edge AI architecture that connects camera sensor nodes to edge devices for real-time image classification.
- Evaluation of the feasibility of local versus offloaded AI inference for vehicle classification in smart cities.
- Quantification of trade-offs among processing performance, dropped frames, and inference latency in single-node deployments.

This paper is organized as follows: Section 2 reviews related work on wireless technologies for smart cities, highlighting trade-offs between 5G and Wi-Fi in urban deployments. Section 3 presents our system's architecture and experimental setup. Section 4 defines the performance metrics used in our evaluation. Section 5 reports and discusses the results of our experiments. Section 6 discusses outcomes and describes challenges still needing to be adressed. Section 7 concludes the paper.

2 RELATED WORK

Understanding the general characteristics of the edge communication and compute infrastructure is essential for supporting smart city applications. Therefore, the work of Chen et al. [4] contributes to understanding the role of Mobile Edge Computing (MEC) in enabling smarter cities. The authors highlight MEC's ability to support latency-sensitive services by placing computation close to the data source, emphasizing leveraging 5G networks to achieve the required bandwidth and low latency. Their architecture envisions MEC nodes at the network edge, enabling applications such as real-time video analytics, augmented reality, and vehicular communication. However, the reliance on 5G infrastructure limits its immediate applicability in areas without widespread 5G coverage or where deployment costs are prohibitive.

Similarly, Garcia et al. [6] analyze wireless technology selection for IoT deployments in smart cities, focusing on how technology choice impacts reliability and performance. They identify overcrowding in the 2.4 GHz ISM band due to the widespread use of Bluetooth, ZigBee, Wi-Fi, mobile communications, beeing a challenge for Worldwide Interoperability for Microwave Access (WiMax), causing interference, packet loss, and degraded performance. Their findings suggest that technology selection must balance application-specific requirements with interference mitigation, making Wi-Fi a viable but context-dependent alternative to cellular-based solutions.

The work of Taleb et al. [21] provides a comprehensive review of smart city architectures, identifying wireless technologies as key enablers of urban IoT services. They recognize both 5G and Wi-Fi as critical in urban deployments, noting that while 5G excels in wide-area, high-mobility scenarios, Wi-Fi offers cost-effective coverage for ultra-local services where infrastructure investment must remain minimal.

Together, these studies emphasize that although 5G promises high performance for large-scale and mobile smart city services, Wi-Fi remains a practical alternative in local or budget-constrained scenarios. Reflecting on the findings by Garcia [6] from 2018, shifting Wi-Fi into the 5/6 Ghz band could resolve many of the problems mentioned. This motivates our exploration of a Wi-Fi-based edge AI framework for real-time traffic monitoring, removing the dependency on wide-area infrastructure while meeting low-latency requirements.

3 FRAMEWORK

In this section, we introduce the framework of a Smart City use case application for performing vehicle classification directly at the roadside, proposing a low-power, low-cost solution, and propagating the information over short range to client applications via Wi-Fi.

As depicted in Figure 1, the architecture of the framework consists of three main components, client, local processing node, and remote inference node, deployed in two different configurations optimized for different objectives, namely **local** and **offloading**. The first deployment, referred to as the **local deployment**, involves a node performing vehicle classification directly on the roadside using a Raspberry Pi 4, which also acquires the input images. This setup uses CPU-only inference to achieve entirely local processing. A client device connects to the local node via Wi-Fi and requests image classification results at regular intervals. The results include annotated vehicle classifications in JSON format and image overlays (see Section 3.2), providing the most recent outcomes to the client applications.

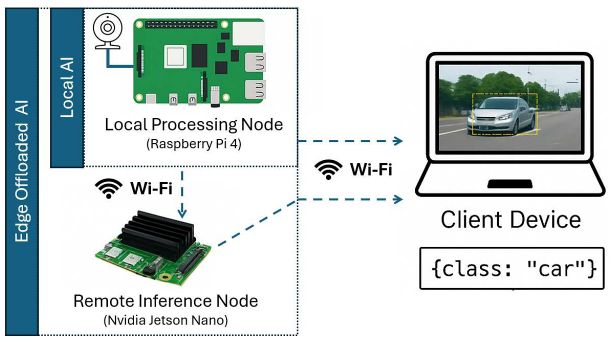


Figure 1: Overview of the system architecture

In the **offloading deployment**, inference is performed remotely on an edge node equipped with a GPU (NVIDIA Jetson Nano) for acceleration. When the client requests it, the local node sends the current image to the remote node, executing the image classification. After processing, the client device retrieves the results from the remote node.

This setup directly compares local (CPU-based) and offloaded (GPU-accelerated) edge processing, quantifying the impact of CPU and GPU inference performance in conjunction with additional networking latency introduced by the offloading process.

3.1 Workflow

This section describes how the three main components interact in the depicted deployment configurations. Based on the deployment configuration, the local processing node conducts image acquisition and inference, where the remote inference node only conducts inference on images provided by the local processing node. The components organize their communication through the following messages:

- **Hello(DeviceId)**: Announces device presence to the client
- **Control**: Comprises multiple subtypes:
  - *ConfigureExperiment*: Declares model, operational mode, and duration of the experiment
  - *ReadyToStart*: Confirms model loaded and warm-up completed
  - *BeginExperiment*: Initiates periodic pulsing
  - *Shutdown*: Performs orderly teardown
- **Pulse(TimingMetadata)**: Emitted by client every Δt
- **Frame**: Conveys image F_i from local to remote node
- **Result**: Returns inference output R_i to client

Both **Frame** and **Result** messages carry the `TimingMetadata` package initially sent by the **Pulse**. This package contains the sequence number i of the current inference cycle, the frame number, and a set of timestamps collected across devices.

The use of individual messages is described in the following sections.

3.1.1 Workflow local inference. is depicted in Figure 2 and initiated by the client in a *Setup*-phase with the local processing node. In the *Setup*-phase, the client awaits `Hello` messages from the local processing node, then issues `Control::ConfigureExperiment` to configure the model, in the current deployment (local or offloading), and defines run parameters (e.g., experiment duration, fixed frame rate). The local processing node loads the model, performs a warm-up cycle, loading the model into memory and executing a dummy inference pass to prepare the runtime, and then replies to the client with `Control::ReadyToStart`.

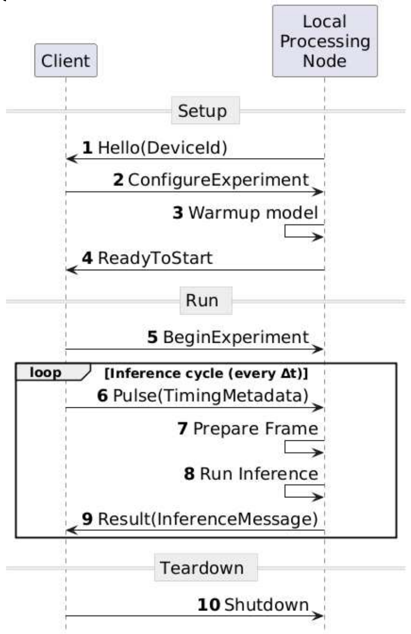


Figure 2: workflow on local deployment configuration using local inference

The *Run*-phase starts with the `Control::BeginExperiment` message indicating a new run. For each inference cycle, the client emits a `Pulse` message carrying a `TimingMetadata` package that includes the current sequence number i. Upon receiving a `Pulse`, the local processing node captures image F_i from the onboard camera. The system packs the captured frame into a `Frame(`F_i`)` message.Upon capturing, the node applies inference and returns the result to the client as `Result(`R_i`)`. The client completes the workflow in the *Teardown* phase by submitting the `Shutdown` message.

3.1.2 Workflow offloaded inference. is depicted in Figure 3, and extends on the **local inference deployment**. During the *Setup*-phase, the client also registers the remote inference node by awaiting `Hello` and then sending the `Control::ConfigureExperiment` message. Both nodes respond with `Control::ReadyToStart`.

During the *Run*-phase, the local processing node transmits `Frame(`F_i`)` to the remote inference node. The remote inference node executes the model to obtain the inference result R_i. The remote node then returns `Result(`R_i`)` to the client. This deployment leverages GPU acceleration at the cost of network transmission overhead, such that the total processing time includes inference and data transfer delays.

The run continues until the configured experiment duration has elapsed, after which the client initiates the *Teardown*-phase by sending `Control::Shutdown` to both nodes, thus terminating the experiment.

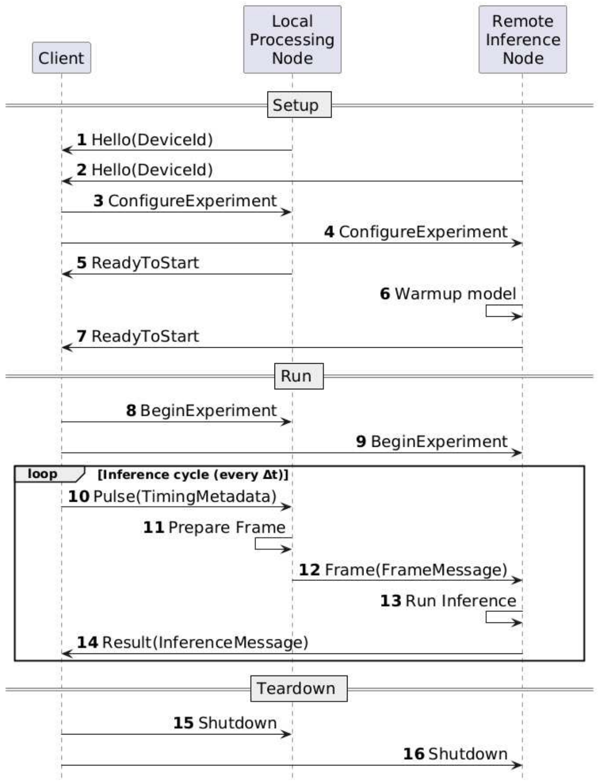


Figure 3: workflow on offloading deployment configuration applying remote inference

3.1.3 Distributed offloading. The proposed architecture is not necessarily constrained to a bijective relationship between the local processing node N^{loc} and remote inference node N^{rem}. Generalizing the queueing concept N^{rem} to support $1 : k$ node relations, where k is the number of $N^{loc} = \{n_0 \dots n_{k-1}\}$ share one N^{rem} node for offloading.

With the capacity queue policy ($\forall n \in N^{loc} : Q_{\text{size}} = 1$), each local processing node supports only the most current frame to be processed.

We must demand the following condition to assign $n_i \in N^{loc}$.

$$\frac{\mu_m}{\sum_{i=0}^{k} \lambda_0(n_i)} \geq 1,$$

where μ_m is the highest sustainable processing rate for model m and λ_0 defines per-node frame generation rate.

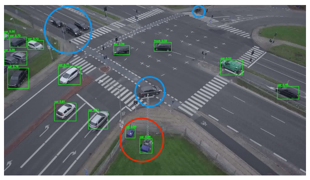
3.2 Vehicle Classification using AI

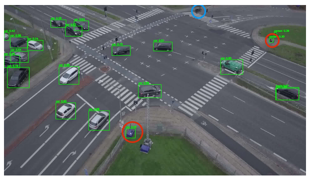
In this section, we describe the purpose of the framework. Many traffic applications widely use these existing AI models for live traffic classification [20] [23] [16]. We also utilize *YOLOv5* [12] as object-detector. The models are trained on the COCO dataset [14, 24] and on car, truck, bus, and motorcycle categories.

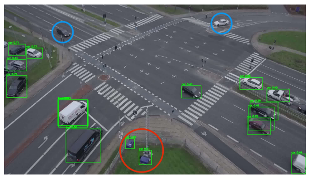
Three model distinguished by their size: `YOLOv5n` (nano), `YOLOv5s` (small), and `YOLOv5m` (medium). `YOLOv5n` variant provides the lowest latency but reduced precision, while `YOLOv5m` achieves higher accuracy at greater computational cost.

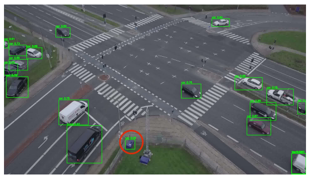
This selection enables systematic evaluation on both the *Local Processing Node* (CPU-only) and the *Remote Inference Node* (GPU-accelerated), showing a trade-off between processing locality. In terms of model size, their parameter counts and corresponding in-memory weights differ substantially, about *7 MB* for the nano-size model, *29 MB* for the medium, and *85 MB* at FP32 precision. The official YOLOv5 model table [12] reports the parameter counts that we use to derive these values, applying the relation of params $\times$ 4 bytes for FP32 storage.

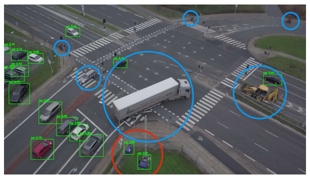
Figure 4 compares the outputs of `YOLOv5n` and `YOLOv5m` on identical frames. Both models detect multiple vehicles, but they also misclassify some objects. For instance, they identify roadside electrical boxes as cars, and in the case of `YOLOv5m`, they also misclassify traffic lights as pedestrians, as highlighted in red in Figures 4a, 4c, and 4e. `YOLOv5n` further fails to detect larger objects such as the truck and excavator (highlighted in blue; Figure 4e), whereas the medium variant succeeds in recognizing them and generally identifies more vehicles overall. These examples illustrate the trade-off between accuracy and model size [17]. Smaller models are computationally efficient but prone to missed detections (Figure 4a), while larger models achieve broader coverage at the expense of higher computational cost (Figure 4f).


(a) Frame 30175; YOLOV5n


(b) Frame 30175; YOLOV5m


(c) Frame 24418; YOLOV5n


(d) Frame 24418; YOLOV5m


(e) Frame 10614; YOLOV5n

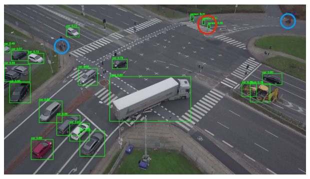

(f) Frame 10614; YOLOV5m

Figure 4: comparison of `YOLOv5n` and `YOLOv5m` on frames from the multi-view traffic intersection dataset [18]

3.3 Implementation

The solution implements a distributed edge computing architecture with four main software components implemented in Rust and Python. The code applies the actual inference in Python. The complete implementation is available on GitHub on our rust-traffic-watch repository[1]

The implementation supports automated benchmarking across multiple AI model variants (YOLOv5n/s/m), configurable frame rates (1-15 frames per second), and supports local- and offloading deployment configurations established in the following software components:

- **Controller**: Orchestrates experiments, coordinates device communication, and collects performance metrics across all distributed nodes
- **Pi Sender**: Captures frames and performs local PyTorch inference or forwards data for offloading to remote compute nodes
- **Jetson Receiver**: Provides accelerated TensorRT inference for offloaded workloads with optimized GPU utilization
- **Shared Library**: Common networking protocols, binary serialization, and type definitions for inter-device communication

All experiments generate detailed CSV logs containing end-to-end latency, processing overhead, network latency, and detection accuracy metrics for reproducible analysis across 24 configurations.

3.3.1 *Hierarchical Time Measurement.*

Accurate time measurements across distributed computing nodes always present difficulties since clocks tend not to be fully synchronized. This paragraph describes the method we use to avoid this problem.

We measure time hierarchically across nodes $\mathcal{N} = \{N_c, N_p, N_r\}$, where N_c is the client, N_p a local processing node, and N_r a remote inference node triggered by N_p. Each node N_i maintains a local clock. The client N_c initializes the ordered timestamp structure $\mathcal{T}$ (`TimingMetadata`):

$$\mathcal{T} = \langle t_1, \ldots, t_n \rangle,$$

During execution, N_p appends timestamps for local processing and triggers N_r, which appends inference timestamps. We compare only timestamps acquired from the same local clock to avoid aligning the local clocks (C_c, C_p, C_r). So we define a duration between to moments τ_a, τ_b on node N_i as

$$\Delta T_i(t_a, t_b) = C_i(\tau_b) - C_i(\tau_a),$$

with $t_a = C_i(\tau_a)$ and $t_b = C_i(\tau_b)$. A valid duration requires:

$$\Delta T_i(t_a, t_b) \quad \text{only if} \quad t_a, t_b \leftarrow C_i,$$

i.e. to calculate the duration of a run is calculated as follows:

$$t_{\text{start}} = C_c(\tau_{\text{start}}),$$
$$t_{\text{end}} = C_c(\tau_{\text{end}}),$$
$$\Delta T_{\text{run}} = t_{\text{end}} - t_{\text{start}}.$$

4 EVALUATION DESIGN

In our evaluation, we systematically vary the capture frame rate *CFR* from 1 to 15 frames per second (FPS) and measure the impact of different inference models on total processing time, accuracy, and system responsiveness. These parameters allow us to analyze trade-offs between computation location, network usage, and classification performance, providing insights into the design of smart

[1] https://github.com/rawalcher/rust-traffic-watch

city edge AI systems, considering the two different implementations of the vehicle classification smart city application.

4.1 Evaluation Metrics

Table 1 summarizes the notation used throughout the evaluation. This table serves as a reference for the various symbols and terms employed in the evaluation.

Table 1: Notation used in describing the experiments

Symbol	Description	Unit
CFR	Capture frame rate	frames/s
DFR	Dropped frame rate	frames/s
OTR	On-time processing rate	frames/s
RPI	Relative performance improvement	-
$\mathcal{Z}_{\text{total}}$	Total nominal frames captured during experiment	frames
$\mathcal{Z}_{\text{dropped}}$	Amount of frames dropped during experiment	frames
ΔT_{total}	Total experiment duration	s
$\Delta T_{\text{proc}}(i)$	Processing time for frame i	s
μ_m	Max. sustainable processing rate for model m	frames/s
λ_0	Per-node frame generation rate	frames/s
Q_{size}	Capacity queue size	frames

4.1.1 *Performance Metrics.*

The target metric in our evaluation to assess image inference is the *capture frame rate* (CFR), defined as:

$$CFR = \frac{\mathcal{Z}_{\text{total}}}{\Delta T_{\text{total}}}.$$

To handle varying processing speeds, we implement a *capacity-bounded queue* with size $Q_{\text{size}} = 1$, ensuring only the most recent frame is processed. When a new frame arrives, it replaces any waiting frames in the queue during ongoing processing. This policy prioritizes the timeline over completeness.

We define the *dropped frame rate* (DFR) as the ratio of frames discarded due to processing delays:

$$DFR = \frac{\mathcal{Z}_{\text{dropped}}}{\mathcal{Z}_{\text{total}}} \times 100\%.$$

A frame is considered dropped when:

(1) It arrives while the previous frame is still being processed $\Delta T_{proc}(i-1) > 1/CFR$, and
(2) It gets replaced by a newer frame before processing begins.

The *on-time processing rate* (OTR) represents successfully processed frames, $OTR = 100\% - DFR$.

4.1.2 *Processing Time.*

The *average processing time per frame* T_{proc} is computed as:

$$\Delta T_{proc} = \frac{1}{n} \sum_{i} \Delta T_{proc}(i).$$

Where n represents the number of frames that the system successfully processes without dropping, while i indicates the index of the current frame, which determines the processing time ΔT_{proc} for that frame, how processing time is cummulated depends on the deployment configuration. $\Delta T_{proc} \in \{\Delta T_{proc}^{\text{loc}}, \Delta T_{proc}^{\text{off}}\}$

For the **local-deployment**, processing time ΔT_{proc}^{loc} accumulates Image acquisition from the camera, model inference, and result serialization.

For the **offloading-deployment** the processing time $\Delta T_{proc}^{\text{off}}$ includes additional components: network transmission t_{in} to the remote node, the queuing delay at the remote node, model inference ΔT_{inf} and network transmission of results ΔT_{out}.

4.1.3 *Performance Comparison.*

To assess the benefit of GPU offloading, we define the *relative performance improvement* (RPI):

$$RPI = \frac{DFR_{\text{local}} - DFR_{\text{offload}}}{DFR_{\text{local}}} \times 100\%.$$

If $RPI > 0$, offloading reduces frame drops, $RPI < 0$ shows that network overhead outweighs acceleration benefits. $RPI = 0$, both approaches work identically.

4.1.4 *Model-Specific Processing Capacity.*

Each model variant $m \in \{\texttt{YOLOv5n}, \texttt{YOLOv5s}, \texttt{YOLOv5m}\}$ exhibits a maximum sustainable processing rate μ_m for each type t processing node and corresponding hardware capabilities. The processing capacity without drops demands:

$$CFR \le \mu_m.$$

Due to the capacity queue implementation, the system can operate at higher capture rates but will drop frames according to:

$$DFR(m) \approx \max\left(0, (1 - \frac{\mu_m}{CFR})\right) \times 100\%.$$

This approximation assumes steady-state operation and uniform processing times.

5 RESULTS

We evaluate two deployment strategies: *local AI* inference directly on the roadside node (Raspberry Pi 4, CPU-only) and *remote inference* offloaded to a Jetson Nano (GPU-enabled) over a Wi-Fi link. Performance is quantified using the metrics introduced in Section 4: the *Dropped Frame Rate* (DFR), the *Capture Frame Rate* (CFR), and the *Inference Latency* (ΔT_{inf}).

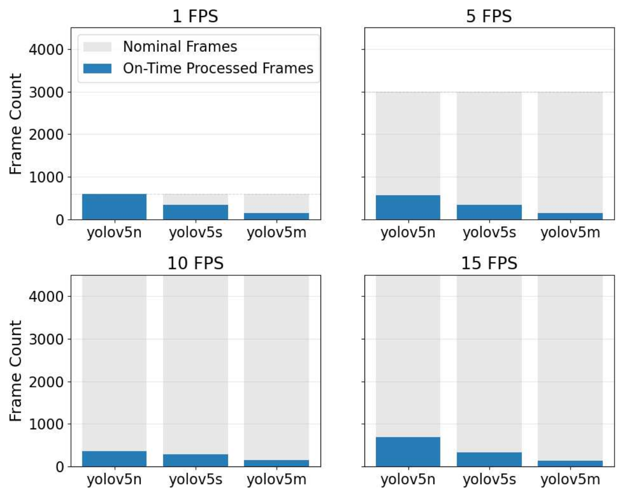


Figure 5: OTR over DFR using local deployment configuration (y-axis limited to 4000 frames)

Reviewing the results of local deployment configuration shows that sustained operation without frame drops ($DFR = 0$) is only achievable with the smallest model, YOLOv5n, at $CFR = 1$. This setting satisfies the condition $OTR \geq CPR$, so the next frame arrives only after the previous one is fully processed. Increasing the capture rate to 5 FPS or 15 FPS results in a significant reduction of successful processings, with DFR exceeding 80% for YOLOv5s and 95% for YOLOv5m. In these cases, the high DFR reflects the condition $OTR \ll CFR$, where most frames cannot be processed in time.

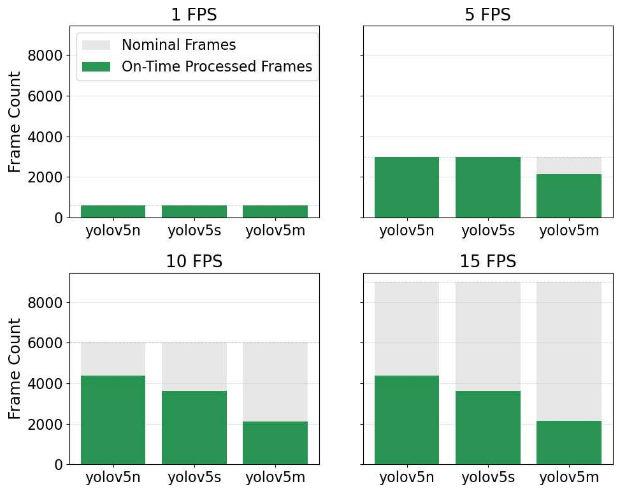


Figure 6: OTR over DFR using offloading deployment configuration

On the other hand, the situation changes quite a bit when offloading. With $CFR = 1$, we stay at $DFR \approx 0$, and we only start dropping frames at $CFR = 5$ with the bigger YOLOv5m model. Starting at $CFR = 10$, all models perform worse across the board with DFR ranging from ≈ 27% (YOLOv5n) up to ≈ 64% (YOLOv5m). At $CFR = 15$, we can see the same results as observed at $CFR = 10$, facing hardware processing limits on the remote inference node.

These findings confirm that for single-node deployments, local inference is feasible only for lightweight models and low capture rates. In contrast, offloading to GPU-equipped edge nodes enables near-real-time performance even for heavier models at higher capture rates, as long as OTR remains sufficiently close to CFR to keep DFR within the acceptable bound DFR^{max}. Furthermore, on lower $CFR \leq 5$ resource capabilities of the offloading node suffice to process data of multiple nodes.

6 DISCUSSION

Based on the experimental evaluation, we found that processing the video stream on a single node can be highly restrictive. Many real-world deployments of edge AI involve multiple processing nodes sharing one offloading node. Such a configuration introduces several open challenges, like resource contention, adaptive scheduling, and reliable network performance despite the actual multi-node distribution of service capabilities.

Multi-node scalability allows us to optimize resource usage. Increasing the number of camera nodes will increase the total data rate and computational demand on the AI node [7], potentially overloading its GPU or saturating the Wi-Fi network (see Section 3.1.3). Understanding the limits of performance degradation and identifying optimal load-balancing and queuing strategies will be critical. This also demands long-term reliability testing in outdoor urban environments with variable interference, integrating more advanced AI models for real-time object detection and tracking, and applying compression techniques to reduce bandwidth requirements without compromising classification accuracy.

Finally, *network optimization* techniques will be necessary to mitigate interference and congestion in dense Wi-Fi environments. This includes channel allocation strategies, use of Wi-Fi 7 [5] features like Multi-Link Operation (MLO) [3] and 4K Quadrature Amplitude Modulation (4K-QAM) [8], and integration with mesh networking to extend coverage without sacrificing performance or going beyond those aspects of 6G.

7 CONCLUSION

This paper introduced a Wi-Fi-based Edge AI framework for real-time traffic image classification in smart city environments. We implemented a system where low-power camera nodes either perform on-device inference using their CPU or offload the processing to an NVIDIA Jetson Nano for GPU-accelerated inference. By defining formal performance metrics such as inference latency and dropped frame rate, we could assess the feasibility of both local and offloaded processing strategies.

Our results demonstrate that purely local processing is only practical for lightweight AI models at low frame rates. For instance, YOLOv5n at 1 FPS processed all frames without drops. However, increasing the frame rate or the model complexity quickly caused dropped frame rates to exceed operational thresholds up to 98% in some configurations. Conversely, GPU offloading to the Jetson Nano maintained adequate processing at low to moderate frame rates across all tested models, with only minimal performance degradation for heavier models at higher frame rates.

These findings underscore the value of GPU-accelerated offloading over Wi-Fi for achieving reliable, low-latency performance in urban traffic monitoring scenarios [22] [15] without the need for costly 5G or wired infrastructure [9]. The approach is particularly attractive for **ultra-local deployments** where coverage is contained within a single Wi-Fi cell, offering a cost-effective yet high-performance alternative to more infrastructure-heavy solutions. Beyond the specific traffic monitoring use case, the proposed framework demonstrates the broader viability of Wi-Fi-based edge computing for other urban sensing applications, provided the computational load and network conditions are appropriately managed. This positions localized Wi-Fi edge AI as a practical enabler for smart city services in settings where traditional wide-area edge or cloud deployments are impractical.

ACKNOWLEDGMENT

This work received funding from the OeAD agency of the Austrian Ministry for Education, Science and Research and the Macedonian Ministry for Education and Science under project number MK10/2024.

REFERENCES

[1] Md Eshrat E Alahi, Arsanchai Sukkuea, Fahmida Wazed Tina, Anindya Nag, Wattanapong Kurdthongmee, Korakot Suwannarat, and Subhas Chandra Mukhopadhyay. 2023. Integration of IoT-enabled technologies and artificial intelligence (AI) for smart city scenario: recent advancements and future trends. *Sensors* 23, 11 (2023), 5206.

[2] Johan Barthélemy, Nicolas Verstaevel, Hugh Forehead, and Pascal Perez. 2019. Edge-computing video analytics for real-time traffic monitoring in a smart city. *Sensors* 19, 9 (2019), 2048.

[3] Marc Carrascosa-Zamacois, Lorenzo Galati-Giordano, Francesc Wilhelmi, Gianluca Fontanesi, Anders Jonsson, Giovanni Geraci, and Boris Bellalta. 2024. Performance Evaluation of MLO for XR Streaming: Can Wi-Fi 7 Meet the Expectations?. In *2024 IEEE 29th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 1–6.

[4] Ning Chen, Tie Qiu, Laiping Zhao, Xiaobo Zhou, and Huansheng Ning. 2021. Edge Intelligent Networking Optimization for Internet of Things in Smart City. *IEEE Wireless Communications* 28, 2 (2021), 26–31. https://doi.org/10.1109/MWC.001.2000243

[5] Cailian Deng, Xuming Fang, Xiao Han, Xianbin Wang, Li Yan, Rong He, Yan Long, and Yuchen Guo. 2020. IEEE 802.11 be Wi-Fi 7: New challenges and opportunities. *IEEE Communications Surveys & Tutorials* 22, 4 (2020), 2136–2166.

[6] Laura García-García, Jose M Jiménez, Miran Taha Abdullah Abdullah, and Jaime Lloret. 2018. Wireless technologies for IoT in smart cities. *Network Protocols and Algorithms* 10, 1 (2018), 23–64.

[7] Raj Hakani and Abhishek Rawat. 2024. Edge computing-driven real-time drone detection using YOLOv9 and NVIDIA Jetson nano. *Drones* 8, 11 (2024), 680.

[8] Roger Pierre Fabris Hoefel. 2024. Effects of Phase Noise and Frequency Offset on the Performance of 4K-QAM and 16K-QAM in 802.11 be and 802.11 bn WLANs. In *2024 IEEE Wireless Communications and Networking Conference (WCNC)*. 1–6.

[9] Kurt Horvath, Dragi Kimovski, Stojan Kitanov, and Radu Prodan. 2025. Enhancing Traffic Safety with AI and 6G: Latency Requirements and Real-Time Threat Detection. In *2025 10th International Conference on Information and Network Technologies (ICINT)*. IEEE, 129–136.

[10] Kurt Horvath, Shpresa Tuda, Blerta Idrizi, Stojan Kitanov, Fisnik Doko, and Dragi Kimovski. 2025. 6G Infrastructures for Edge AI: An Analytical Perspective. *arXiv preprint arXiv:2506.10570* (2025).

[11] Monica Karel Huerta, Jessica Garizurieta, Rubén González, Luis-Ángel Infante, Melina Horna, Renato Rivera, and Roger Clotet. 2023. A long-distance wifi network as a tool to promote social inclusion in southern veracruz, mexico. *Sustainability* 15, 13 (2023), 9939.

[12] Glenn Jocher, Ayush Chaurasia, Jirka Borovec, and Ultralytics. 2020. YOLOv5. https://github.com/ultralytics/yolov5.

[13] Dragi Kimovski, Roland Mathá, Josef Hammer, Narges Mehran, Hermann Hellwagner, and Radu Prodan. 2021. Cloud, Fog, or Edge: Where to Compute? *IEEE Internet Computing* 25, 4 (2021), 30–36. https://doi.org/10.1109/MIC.2021.3050613

[14] Zhiming Luo, Frederic Branchaud-Charron, Carl Lemaire, Janusz Konrad, Shaozi Li, Akshaya Mishra, Andrew Achkar, Justin Eichel, and Pierre-Marc Jodoin. 2018. MIO-TCD: A new benchmark dataset for vehicle classification and localization. *IEEE Transactions on Image Processing* 27, 10 (2018), 5129–5141.

[15] Aale Luusua, Johanna Ylipulli, Marcus Foth, and Alessandro Aurigi. 2023. Urban AI: Understanding the emerging role of artificial intelligence in smart cities. *AI & society* 38, 3 (2023), 1039–1044.

[16] Madhusri Maity, Sriparna Banerjee, and Sheli Sinha Chaudhuri. 2021. Faster r-cnn and yolo based vehicle detection: A survey. In *2021 5th international conference on computing methodologies and communication (ICCMC)*. IEEE, 1442–1447.

[17] Gaurav Menghani. 2023. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *Comput. Surveys* 55, 12 (2023), 1–37.

[18] Andreas Møgelmose. 2019. Multi-view Traffic Intersection Dataset. Kaggle. https://www.kaggle.com/datasets/andreasmoegelmose/multiview-traffic-intersection-dataset

[19] Zhaolong Ning, Jun Huang, and Xiaojie Wang. 2019. Vehicular fog computing: Enabling real-time traffic management for smart cities. *IEEE Wireless Communications* 26, 1 (2019), 87–93.

[20] Héctor Rodríguez-Rangel, Luis Alberto Morales-Rosales, Rafael Imperial-Rojo, Mario Alberto Roman-Garay, Gloria Ekaterine Peralta-Peñuñuri, and Mariana Lobato-Báez. 2022. Analysis of statistical and artificial intelligence algorithms for real-time speed estimation based on vehicle detection with YOLO. *Applied Sciences* 12, 6 (2022), 2907.

[21] Tarik Taleb, Sunny Dutta, Adlen Ksentini, Muddesar Iqbal, and Hannu Flinck. 2017. Mobile Edge Computing Potential in Making Cities Smarter. *IEEE Communications Magazine* 55, 3 (2017), 38–43. https://doi.org/10.1109/MCOM.2017.1600249CM

[22] Radosław Wolniak and Kinga Stecuła. 2024. Artificial intelligence in smart cities—applications, barriers, and future directions: a review. *Smart cities* 7, 3 (2024), 1346–1389.

[23] Yu Zhang, Zhongyin Guo, Jianqing Wu, Yuan Tian, Haotian Tang, and Xinming Guo. 2022. Real-time vehicle detection based on improved yolo v5. *Sustainability* 14, 19 (2022), 12274.

[24] Muhammad Azhad Bin Zuraimi and Fadhlan Hafizhelmi Kamaru Zaman. 2021. Vehicle detection and tracking using YOLO and DeepSORT. In *2021 IEEE 11th IEEE symposium on computer applications & industrial electronics (ISCAIE)*. IEEE, 23–29.

Predicting Encoding Energy from Low-Pass Anchors for Green Video Streaming

Zoha Azimi
Institute of Information Technology, University of Klagenfurt
Klagenfurt, Austria

Reza Farahani
Institute of Information Technology, University of Klagenfurt
Klagenfurt, Austria

Vignesh V Menon
Video Communication and Applications Dept, Fraunhofer HHI
Berlin, Germany

Christian Timmerer
Institute of Information Technology, University of Klagenfurt
Klagenfurt, Austria

Abstract

Video streaming now represents the dominant share of Internet traffic, as ever-higher-resolution content is distributed across a growing range of heterogeneous devices to sustain user Quality of Experience (QoE). However, this trend raises significant concerns about energy efficiency and carbon emissions, requiring methods to provide a trade-off between energy and QoE. This paper proposes a lightweight energy prediction method that estimates the energy consumption of high-resolution video encodings using reference encodings generated at lower resolutions (so-called anchors), eliminating the need for exhaustive per-segment energy measurements, a process that is infeasible at scale. We automatically select encoding parameters, such as resolution and quantization parameter (QP), to achieve substantial energy savings while maintaining perceptual quality, as measured by the Video Multimethod Fusion Assessment (VMAF), within acceptable limits. We implement and evaluate our approach with the open-source VVenC encoder on 100 video sequences from the Inter4K dataset across multiple encoding settings. Results show that, for an average VMAF score reduction of only 1.68, which stays below the Just Noticeable Difference (JND) threshold, our method achieves 51.22 % encoding energy savings and 53.54 % decoding energy savings compared to a scenario with no quality degradation.

CCS Concepts

• **Information systems → Multimedia streaming**; • **Computing methodologies → Artificial intelligence**.

Keywords

Video Streaming, Video on Demand, Machine Learning, Energy Efficiency.

1 Introduction

Video streaming applications, such as live content and Video-on-Demand (VoD), now dominate global Internet traffic as recent reports indicate that video content accounts for over 70 % of total traffic today, with projections exceeding 80 % by 2028 [1]. HTTP Adaptive Streaming (HAS) methods such as MPEG Dynamic Adaptive Streaming over HTTP (DASH) [2, 3] and Apple HTTP Live Streaming (HLS) [4] have become the de facto standard video delivery method. In these methods, each video is encoded into multiple resolution–bitrate pairs, forming a bitrate ladder, from which clients dynamically select the most suitable representation according to current network and device conditions [5]. However, constructing such ladders requires encoding each video sequence into multiple representations, a process that is both computationally intensive and energy-demanding [6]. This energy cost is further intensified by modern video codecs such as High Efficiency Video Coding (HEVC) [7] and Versatile Video Coding (VVC) [8], which achieve higher compression efficiency through advanced prediction, partitioning, and transform coding tools [9, 10].

This highlights a key challenge in adaptive streaming, i.e., balancing video quality, compression efficiency, and energy consumption, where the choice of the proper encoding configuration plays a pivotal role in achieving this balance [11, 12]. Parameters such as resolution, framerate, and quantization parameter (QP) directly influence compression efficiency, perceptual quality, and energy consumed during encoding and decoding [11, 13, 14]. Since higher video quality levels typically require higher energy costs, efficient configurations become essential for sustainable video streaming. While heuristic [15] or Artificial Intelligence (AI)-driven methods [16–18] have been proposed to optimize configuration selection, they primarily focus on compression efficiency and perceptual quality, giving insufficient attention to energy considerations. Moreover, accurate energy evaluation requires per-segment energy measurements across multiple configurations, which is infeasible at scale.

This paper proposes a practical and scalable scheme that uses reference encodings generated at lower resolutions (hereafter referred to as anchors) as a proxy to predict the energy consumption of high-resolution representations. The core hypothesis is that encoding time and energy are strongly correlated, allowing patterns observed in low-resolution encodings to be leveraged for predicting the energy required at higher resolutions. For example, encoding a sequence at 360p or 540p resolution with a fixed QP typically completes much faster and consumes less energy than its 1080p or 2160p counterparts, yet still captures the content's inherent characteristics such as motion complexity, texture richness, and scene dynamics. Using these anchors, we train machine learning (ML) models to predict higher-resolution energy consumption without exhaustive measurements, guiding an energy-aware configuration strategy that minimizes energy use while preserving perceptual quality, evaluated through Signal-to-Noise Ratio (PSNR) [19] and Video Multimethod Fusion Assessment (VMAF) [20]. The main contributions of this paper are as follows:

- **Dataset generation**: We construct a dataset of encoding and decoding time, energy consumption, and PSNR, VMAF scores for 100 video sequences from the Inter4K [21] dataset.
- **Anchor-based modeling**: We introduce the concept of low-resolution anchor encodings and demonstrate that their measurements provide meaningful insights into energy consumption trends at higher resolutions.
- **ML-based predictions**: We develop ML models that leverage features extracted from anchor encodings to accurately estimate both energy consumption and perceptual quality.
- **Energy-aware configuration strategy**: We design an encoding parameter selection method that uses predicted energy and quality values, minimizing energy consumption while maintaining visual quality.

2 Related work

2.1 Energy consumption prediction

Several works have proposed methods to estimate the energy consumption during video encoding or decoding. Ghasempour *et al.* [22] proposed a lookup table method for the fast estimation of encoding and decoding energy based on video content, resolution, and framerate features. Sharrab *et al.* [23] introduced a linear regression model for encoding energy consumption using the motion estimation range of video and QP. Azimi *et al.* [13] developed Extreme Gradient Boosting (XGBoost) models to estimate encoding energy consumption using features such as video complexity, quantization parameter (QP), resolution, frame rate, and codec type. Herglotz *et al.* [24] used linear regression models to estimate decoding energy based on decoding processing time. Turkkan *et al.* [25] developed a neural network-based model to predict the decoding power consumption of a video sequence using parameters such as bitrate, resolution, framerate, and file size. Farahani *et al.* [26] introduced a relative decoding energy index (RDEI), a metric that normalizes decoding energy consumption against a baseline encoding configuration, enabling cross-platform comparability and guiding energy-efficient streaming adaptations.

State-of-the-art limitations: These methods rely on full encodings or decodings with specialized energy measurement tools, making them resource-intensive and limited in their ability to represent all encoding scenarios.

2.2 Encoding parameter configuration

Another line of research focuses on selecting encoding parameters to optimize video quality and efficiency. Lebreton *et al.* [27] designed bitrate ladders based on user quitting probabilities, improving the perceived Quality of Experience (QoE). In parallel, ML-based approaches, such as Random Forest (RF)–based models [28], have been employed to predict optimal segment resolutions for enhanced perceptual quality. Huang *et al.* [17] proposed a reinforcement learning-based method to dynamically select bitrate-resolution pairs, jointly optimizing video quality, storage cost, and adaptation to network conditions. Azimi *et al.* [29] used XGBoost models to predict the

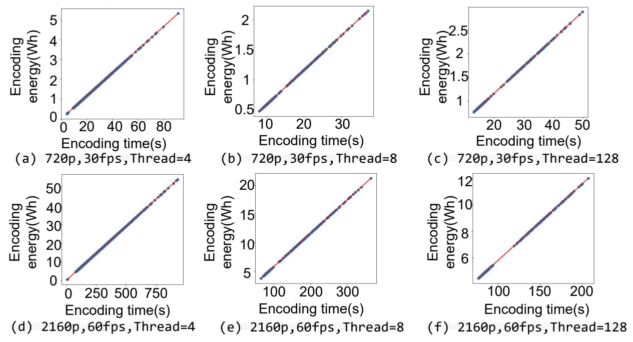


Figure 1: The correlation between encoding time and encoding energy for 100 video sequences, encoded with 720p/30fps and 2160p/60fps with three different number of threads.

decoding time and used a decoding time-constrained configuration setting. Rajendran *et al.* [16] used Pareto-front analysis to predict optimized framerates, constructing decoding complexity-aware ladders. Similarly, Katsenou *et al.* [14] used video quality, decoding time, and bitrate to optimize bitrate ladder construction.

State-of-the-art limitations: While these approaches advance perceptual quality and compression efficiency, they largely neglect energy consumption, an increasingly critical factor for sustainable video streaming.

3 Motivation

Our approach is motivated by two key observations from preliminary experiments, enabling efficient energy prediction across multiple encoding settings.

First, Fig. 1 shows the correlation between encoding time and energy consumption across 100 video sequences from the Inter4K dataset [21] encoded at 720p/30fps and 2160p/60fps. The experiments were conducted using different numbers of threads (4, 8, and 128). In all configurations, a strong linear correlation between encoding time and energy consumption is observed. Similar strong correlations have been observed in other works [24]. Unlike energy measurement, measuring execution time is computationally inexpensive and does not require specialized instrumentation. Thus, we use encoding time as a reliable and practical proxy for energy consumption.

Second, we observe that encoding times across different representations of the same video are strongly correlated. Fig 2 depicts the relationship between average encoding time (in seconds, shown on a logarithmic scale) and the average correlation of encoding times across different video representations (resolutions and QPs) across our 100 test video sequences. For each resolution-QP pair (e.g., 360p, QP47), we computed the correlation between its encoding times and those of all other pairs across 100 video sequences. Overall, average pairwise correlations exceed 0.65, revealing consistent temporal behavior across representations, with the correlation peaks around 0.8 for medium encoding settings (720p–1080p, QP:27–37). To exploit this relationship, we select the representation with the lowest resolution and highest QP as the anchor value (i.e., red star) for each video sequence, as it provides the fastest encoding with

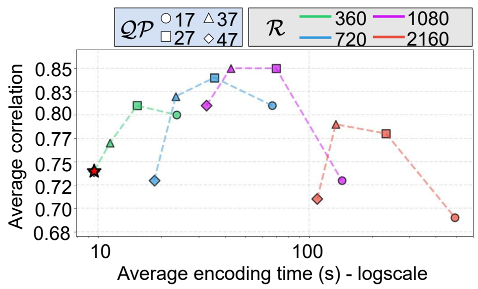


Figure 2: Average correlation of encoding times across 100 video sequences for different resolutions and QPs. Each point shows the mean correlation of one configuration with all others, plotted against its average encoding time (log scale).

minimal computational cost, achieving a correlation of 0.74 while requiring only 9.58 s of encoding time. Thus, by measuring the anchor's encoding time, we can infer the energy consumption of all other representations, significantly reducing computation and energy costs. This concept is extended to predict decoding energy and quality metrics using the anchor's decoding time and perceptual quality measurements, respectively.

4 System Design

4.1 Architecture

Fig. 3 shows the proposed architecture with three main components:

(i) Anchor processing encodes and decodes the lowest resolution (r_{min}) and highest quantization parameter (qp_{max}) representation as the anchor, measures its encoding time (t^A_{enc}), decoding time (t^A_{dec}), and quality metric (q^A), such as PSNR or VMAF.

(ii) Prediction module takes the anchor processing outputs (t^A_{enc}, t^A_{dec}, q^A) together with the target resolutions ($\mathcal{R}$) and QPs ($\mathcal{QP}$) as input and employs f_{enc}, f_{dec}, and f_q to predict the encoding energy ($\hat{e}_{enc}$), decoding energy ($\hat{e}_{dec}$), and quality ($\hat{q}$) for the target video representations.

$$\hat{e}_{enc} = f_{enc}(t^A_{enc}, r \in \mathcal{R}, qp \in \mathcal{QP}),$$
$$\hat{e}_{dec} = f_{dec}(t^A_{dec}, r \in \mathcal{R}, qp \in \mathcal{QP}),$$
$$\hat{q} = f_q(q^A, r \in \mathcal{R}, qp \in \mathcal{QP}).$$

(iii) Green encoding configurations leverages the prediction results to recommend encoding parameters (r, qp) based on an acceptable quality degradation factor ρ. When $\rho = 0$, no quality degradation is allowed, and the encoding parameters are selected to provide the highest visual quality. In contrast, when $\rho = 1$, the configurations prioritize minimum energy consumption, regardless of quality.

4.2 Execution workflow

Algo. 1 outlines the execution workflow of our proposed method. For a given video sequence, a set of target resolutions $\mathcal{R}$ and quantization parameters $\mathcal{QP}$ are defined, along with an acceptable quality degradation factor ρ. First, the video sequence is encoded at the lowest resolution (r_{min}) and highest QP (qp_{max}), which serves as an anchor. The anchor's encoding time (t^A_{enc}), decoding time (t^A_{dec}),

Algorithm 1: Proposed Green Encoding Framework

```
   Input: $\mathcal{R}$, $\mathcal{QP}$, $\rho \in [0,1]$
   Output: Selected representation $(r^*, qp^*)$
   // Step 1: Anchor Processing
1  $t^A_{enc} \leftarrow Encode(r_{min}, qp_{max})$
2  $t^A_{dec}, q^A \leftarrow Decode(r_{min}, qp_{max})$
   // Step 2: Prediction Module
3  $\hat{E}_{enc} \leftarrow [\,], \hat{E}_{dec} \leftarrow [\,], \hat{Q} \leftarrow [\,]$
4  for $r \in \mathcal{R}$ do
5      for $qp \in \mathcal{QP}$ do
6          $\hat{e}_{enc} \leftarrow f_{enc}(t^A_{enc}, r, qp)$
7          $\hat{e}_{dec} \leftarrow f_{dec}(t^A_{dec}, r, qp)$
8          $\hat{q} \leftarrow f_q(q^A, r, qp)$
9          $\hat{E} \leftarrow \hat{e}_{enc} + \hat{e}_{dec}$
10         $\hat{Q} \leftarrow \hat{q}$
   // Step 3: Green Configuration Selection
11 $\hat{q}_{max} \leftarrow Max(\hat{Q})$
12 $\mathcal{F} \leftarrow \{(r, qp) \mid \hat{Q} \geq (1-\rho) \cdot \hat{q}_{max}\}$
13 $(r^*, qp^*) \leftarrow \arg\min_{(r,qp) \in \mathcal{F}} \hat{E}$
14 return $(r^*, qp^*)$
```

and quality metric (q^A) are then measured (lines 1–2). Next, for each resolution $r \in \mathcal{R}$ (line 4) and quantization parameter $qp \in \mathcal{QP}$ (line 5), the encoding energy prediction model f_{enc} (line 6), the decoding energy prediction model f_{dec} (line 7), and the quality prediction model f_q (line 8) are invoked, yielding the predicted encoding energy ($\hat{e}_{enc}$), decoding energy ($\hat{e}_{dec}$), and quality ($\hat{q}$) for each representation. The predicted energies are then aggregated into $\hat{E}$ (line 9), while the quality predictions are stored in $\hat{Q}$ (line 10). After computing the maximum obtainable quality $\hat{q}_{max}$ (line 11), all representations whose predicted quality falls within the acceptable threshold defined by ρ are identified (line 12), and the one with the lowest predicted energy consumption is selected (lines 12–13). Finally, the representation (r^*, qp^*) that satisfies the quality constraint and minimizes energy consumption is returned (line 14). The time complexity of Algo. 1 is $O(|\mathcal{R}| \times |\mathcal{QP}|)$, where $|\mathcal{R}|$ and $|\mathcal{QP}|$ denote the number of target resolutions and quantization parameters, respectively (i.e., the number of target representations of the bitrate ladder).

5 Evaluation Setup

We conducted all experiments on a server with a 128-core Intel Xeon Gold CPU and two NVIDIA Quadro GV100 GPUs. The following subsections describe dataset characteristics and analysis, ML-based prediction models, and evaluation metrics.

5.1 Dataset analysis

We used 100 ultra-high-definition (UHD) video sequences with diverse spatiotemporal characteristics from the Inter4K dataset [21]. To verify that the selected subset is representative of the full Inter4K dataset, which contains 1000 sequences, we applied a Self-Organizing Map (SOM) [30] clustering on the content-complexity features (E_Y, h, L_Y) using the Video Complexity Analyzer (VCA) tool [31] for both the full dataset and our subset. As shown in Fig. 4, the selected subset spans all clusters, confirming that it is representative of the overall dataset in terms of video complexity. Table 1 reports the statistical distribution of the content-complexity

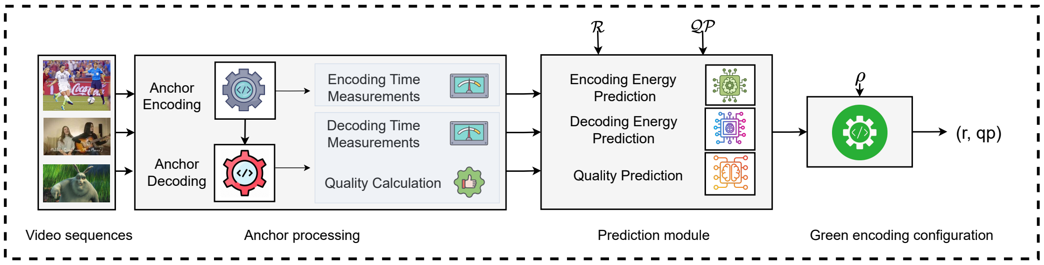


Figure 3: Proposed system architecture.

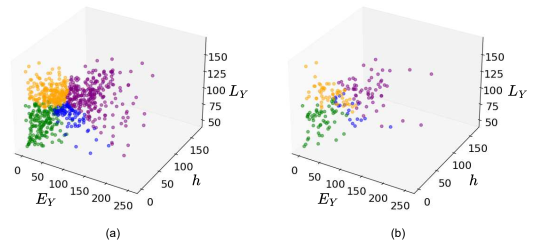


Figure 4: SOM-based clustering on the video complexity features on (a) the full dataset, and (b) our 100 subset.

Table 1: Statistical distribution of E_Y, h, and L_Y across SOM clusters for the full dataset and our subset.

Cl.	Set	E_Y			h			L_Y		
		Avg.	min	max	Avg.	min	max	Avg.	min	max
0 (Purple)	Full	107.4	37.4	252.4	59.4	10.4	171.7	116.3	59.9	138.9
	Sub.	98.8	37.4	252.4	66.1	18.9	165.8	118.9	97.8	138.6
1 (Orange)	Full	41.1	2.7	90.4	25.4	1.9	72.6	123.7	99.2	166.9
	Sub.	48.6	8.8	113.6	20.5	2.6	45.3	125.2	111.0	166.9
2 (Blue)	Full	87.4	48.2	174.9	22.1	2.8	78.8	100.7	52.9	122.2
	Sub.	70.8	23.3	168.2	44.3	15.1	68.5	94.8	72.5	113.9
3 (Green)	Full	29.0	0.9	84.7	15.3	0.5	59.6	83.2	47.4	111.5
	Sub.	31.6	1.0	92.2	14.4	0.5	44.7	87.2	47.4	108.9

features (E_Y, h, L_Y) across the SOM clusters for both the full dataset and our selected subset. While minor variations exist in individual cluster values (e.g., higher h and lower E_Y in Cluster 2 for the subset), the overall ranges and mean values remain consistent.

We encoded each video sequence at 60 fps using `VVenC v1.11` [32] encoder with the `faster` preset [33]. The encoding configuration includes resolutions $\mathcal{R} = \{360, 540, 720, 1080, 1440, 2160\}$p and quantization parameters $\mathcal{QP} = \{17, 22, 27, 32, 37, 42, 47\}$ [34]. The decoding process was applied using `VVdeC v2.3.0` [35]. We recorded the encoding time, encoding energy consumption, decoding time, decoding energy consumption as well as quality scores measured by PSNR and VMAF. The energy consumption was measured with the CodeCarbon tool [36], which tracks the energy consumption of the underlying hardware using Running Average Power Limit (RAPL) [37] for the CPU and nvidia-ml-py [38] for the GPU.

Fig. 5 presents the impact of different $\mathcal{R}$ on encoding and decoding energy, bitrate, PSNR, and VMAF. As expected, higher resolutions substantially increase both encoding and decoding energy, while improving video quality. Fig. 6 shows the variations in encoding and decoding time, bitrate, PSNR, and VMAF across different $\mathcal{QP}$ levels. As QP increases, encoding and decoding time and bitrate decrease, while quality metrics deteriorate accordingly.

5.2 Prediction models

We evaluated six well-known ML models covering four different categories: (1) *Linear Regression* (LR) [39] and *Ridge Regression* (Ridge) [40] as linear models; (2) *Random Forest* (RF) [41] as a tree-based ensemble model; (3) *XGBoost* (XGB) [42] and *LightGBM* (LGBM) [43] as gradient boosting-based ensembles; (4) *Multi-Layer Perceptron* (MLP) [44], a neural network with fully connected layers.

We partitioned the dataset into training (70 %) and testing (30 %) sets at the video level, ensuring that all segments from a given video belong exclusively to one set, avoiding data leakage. We randomly shuffled video identifiers with a fixed seed, guaranteeing reproducibility and consistent splitting across the three prediction tasks. We applied GridSearchCV [45] from Scikit-learn [46] with five-fold cross-validation on the training set. We performed an exhaustive grid search over predefined hyperparameter spaces as summarized in Table 2. While the LR model has no tunable parameters, the Ridge model requires optimization of the regularization parameter α. Tree-based and gradient boosting models required optimization of the number of estimators (n_{trees}), maximum tree depth ($d_{\max}$), and learning rate (η), along with model-specific parameters such as subsampling rates or the number of leaves. For the MLP model, we fine-tuned the number (h_{num}) and size of hidden units (h_{size}) and learning rate (η).

5.3 Evaluation metrics

We use the following metrics to evaluate the accuracy and generalization performance of the prediction models: *Coefficient of determination (R^2)* measures the proportion of variance in the ground truth explained by the model; a higher R^2 indicates better predictive accuracy. *Mean absolute error (MAE)* measures the average relative prediction error; lower MAE indicates higher accuracy. *Root mean squared error (RMSE)* represents the square root of the average squared prediction error, penalizing large deviations more heavily; lower RMSE indicates better performance. *Standard deviation of absolute errors (SDAE)* measures the variability of absolute prediction errors; lower SDAE indicates more consistent predictions.

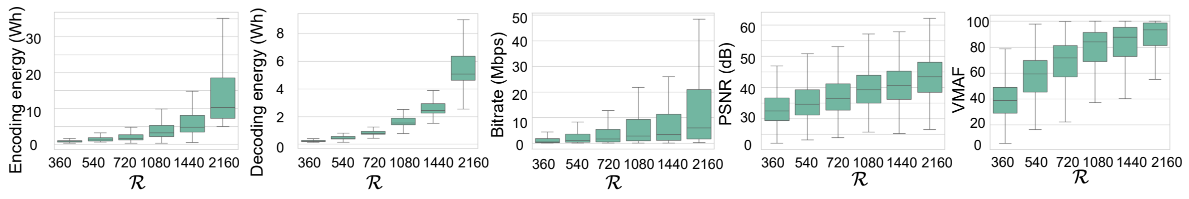


Figure 5: Impact of resolution variations on encoding and decoding energy, bitrate, PSNR, and VMAF across 100 video sequences.

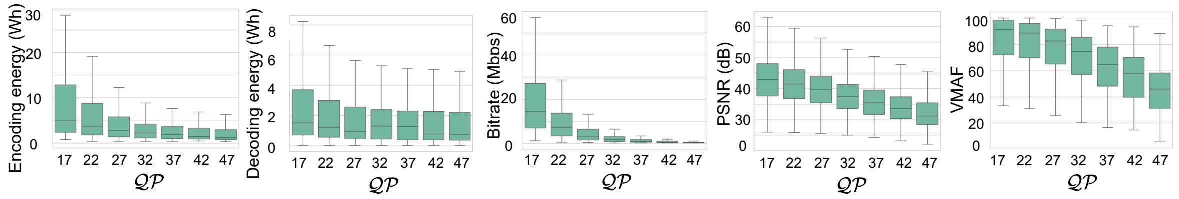


Figure 6: Impact of QP variations on encoding and decoding energy, bitrate, PSNR, and VMAF across 100 video sequences.

Table 2: Hyperparameter search space explored for ML-based prediction models.

Predictive model	*Explored hyperparameters*
LR	None
Ridge	$\alpha \in \{0.1, 1.0, 10.0, 100.0\}$
RF	$n_{trees} \in \{50, 100, 200\}$, $d_{max} \in \{\text{None}, 10, 20\}$ $min\_samples\_split \in \{2, 5\}$ $min\_samples\_leaf \in \{1, 2\}$
XGBoost	$n_{trees} \in \{50, 100, 200\}$, $d_{max} \in \{3, 6, 9\}$ $\eta \in \{0.01, 0.1, 0.2\}$, $subsample \in \{0.8, 1.0\}$
LightGBM	$n_{trees} \in \{50, 100, 200\}$, $d_{max} \in \{3, 6, 9\}$ $\eta \in \{0.01, 0.1, 0.2\}$, $num\_leaves \in \{31, 50, 100\}$
MLP	$h_{size} \in \{64, 128, 256\}, h_{num} \in \{1, 2\}$ $\eta \in \{0.001, 0.01\}$

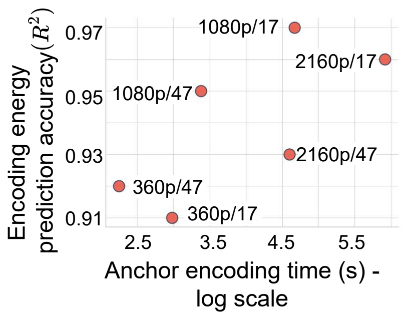


Figure 7: Encoding energy prediction accuracy using different anchors.

We also assess the green encoding configuration module via: *Energy savings* quantifies the reduction in average encoding and decoding energy consumption (Wh) compared to the highest-quality scenario ($\rho = 0$). *Average quality* reports the average PSNR (dB) and VMAF scores across different encoding scenarios (varying ρ). *Average quality drop* measures the reduction in PSNR (dB) and VMAF relative to the highest-quality scenario ($\rho = 0$).

6 Evaluation Results

6.1 Anchor selection analysis

Fig. 7 shows the accuracy of energy prediction models, when a different resolution-QP pair was selected as an anchor. We evaluated six resolution–QP pairs to cover low (360p), medium (1080p), and high (2160p) resolutions, with the minimum (QP = 17) and maximum (QP = 47). The x-axis shows the average encoding time required for each anchor, averaged across 100 video sequences. The results show that higher R^2 values are obtained at the expense of substantially longer encoding times (up to 372.48 s for 2160p/17), whereas the fastest configuration (9.48 s for 360p/47) still achieves reasonable accuracy (R^2 = 0.92). These findings validate the rationale discussed in Section 3, confirming that the optimal anchor corresponds to the configuration with the lowest encoding time.

6.2 Prediction models analysis

We evaluated the performance of the candidate prediction models on four target metrics: encoding energy, decoding energy, PSNR, and VMAF. The reported results represent the average values across the entire test set.

(1) Encoding energy prediction. Table 3 (Encoding Energy) shows that MLP achieved the highest R^2 (0.91) and the lowest RMSE among all models, with $h_{num} = 1, h_{size} = 64$, and $\eta = 0.01$.
(2) Decoding energy prediction Table 3 (Decoding Energy) shows that RF, XGB, LGBM, and MLP have similar performance with R^2 (0.95). We selected LGBM due to its slightly lower MAE (0.01). LGBM achieved its best performance with $n_{leaves} = 50$, $n_{estimators} = 50$, $d_{\max} = 3$ and $\eta = 0.1$.
(3) PSNR prediction Table 3 (PSNR) shows that MLP and LGBM achieved the best overall performance, with the highest R^2 (0.92) among all models. However, with slightly lower RMSE (1.93) and MAE (1.33), MLP with $h_{num} = 2, h_{size} = 256, 128$ and $\eta = 0.001$ was selected for PSNR prediction.

Table 3: Prediction results for encoding energy, decoding energy, PSNR, and VMAF across all models.

Model	Encoding Energy				Decoding Energy				PSNR				VMAF			
	$R^2 \uparrow$	RMSE ↓	MAE ↓	SDAE ↓	$R^2 \uparrow$	RMSE ↓	MAE ↓	SDAE ↓	$R^2 \uparrow$	RMSE ↓	MAE ↓	SDAE ↓	$R^2 \uparrow$	RMSE ↓	MAE ↓	SDAE ↓
LR	0.85	2.39	1.06	2.14	0.91	0.06	0.03	0.04	0.86	2.56	1.94	1.67	0.64	13.15	10.82	7.47
Ridge	0.84	2.39	1.06	2.14	0.91	0.06	0.03	0.04	0.86	2.54	1.93	1.64	0.66	12.74	10.67	6.95
RF	0.89	1.99	0.79	1.83	0.95	0.04	0.02	0.04	0.91	1.98	1.40	1.40	0.89	7.09	4.97	5.06
XGB	0.90	1.89	0.70	1.75	0.95	0.04	0.02	0.03	0.91	2.03	1.40	1.40	0.90	6.82	5.08	4.55
LGBM	0.90	1.88	0.74	1.73	**0.95**	**0.04**	**0.01**	**0.03**	0.92	1.94	1.36	1.37	0.90	6.77	5.04	4.52
MLP	**0.91**	**1.82**	**0.71**	**1.67**	0.95	0.04	0.02	0.04	**0.92**	**1.93**	**1.33**	**1.40**	**0.92**	**6.40**	**4.78**	**4.26**

Table 4: Average VMAF and PSNR drops, and encoding, decoding energy savings for different ρ values.

ρ	Avg. VMAF ↑	Avg. PSNR ↑	VMAF drop ↓	PSNR drop ↓	Enc. energy savings% ↑	Dec. energy savings% ↑
0	99.74	50.05	0	0	0	0
0.05	98.06	46.38	1.68	3.67	51.22	53.54
0.1	94.74	44.16	5	5.89	70.75	70.28
0.3	72.97	37.57	26.77	12.48	92.64	89.90
0.5	53.06	33.97	46.68	16.07	95.82	94.25
0.7	34.83	31.30	64.91	18.74	97.32	96.88
1	31.52	30.85	68.22	19.19	97.58	97.42

(4) VMAF prediction Table 3 (VMAF) shows that MLP achieved the best overall performance, achieving the lowest RMSE (6.4) and MAE (4.78) and highest R^2 (0.92) among all models with $h_{num} = 2, h_{size} = 256, 128$ and $\eta = 0.001$ and therefore was selected for VMAF prediction.

6.3 Green encoding configuration

Table 4 reports the average energy savings (encoding and decoding) and quality degradation (PSNR, VMAF) across the test video sequences for different ρ values, using the configuration with $\rho = 0$ (no quality degradation) as the reference. We also report the average quality scores (PSNR and VMAF) corresponding to the selected encoding configurations.

For $\rho = 0.05$, i.e., allowing a 5 % degradation in VMAF, the average quality loss remains minimal (1.68 VMAF points and 3.67dB PSNR) while achieving substantial energy saving of 51.22 % in encoding and 53.54 % in decoding. The minimum noticeable quality difference, referred to as the just-noticeable-difference (JND) for VMAF, has been reported to range between 2 and 6 in prior works [47–49], indicating that the observed quality loss is likely imperceptible. As ρ increases, the potential energy savings grow further but at the expense of more noticeable quality degradation. For instance, at $\rho = 0.3$, energy savings exceed 90 %, whereas perceived quality declines by more than 25 VMAF points. These results indicate that a modest quality relaxation (e.g., $\rho = 0.05$) yields substantial energy savings while having a negligible impact on perceived visual quality.

Fig. 8 illustrates the impact of $\rho = 0.05, 0.1$ values on four video sequences with different complexity levels. Each plot shows the changes in encoding energy consumption and VMAF across resolutions. The QP is fixed to allow visualization in a two-dimensional plane. The selected resolution differs for each sequence due to variations in content complexity, such as color dynamics, scene change frequency, motion intensity, and brightness. For example, under $\rho = 0.05$, sequence 72 (Fig. 8 (a)) at 1440p resolution achieves a 50.4 % energy saving while maintaining acceptable quality compared to 2160p. Sequence 65 (Fig. 8 (b)) stays within 5 % quality degradation at 1080p. Even when the degradation threshold is increased to 10 % ($\rho = 0.1$), the selected resolution remains 1080p, as lower resolutions would exceed the allowed quality loss. For sequence 23 (Fig. 8 (c)), the selected resolutions are 1080p and 720p for $\rho = 0.05$ and 0.1, respectively. In contrast, for sequence 98 (Fig. 8 (d)), a lower resolution of 720p is sufficient to meet the quality constraint of $\rho = 0.05$, resulting in an 81.1 % energy saving.

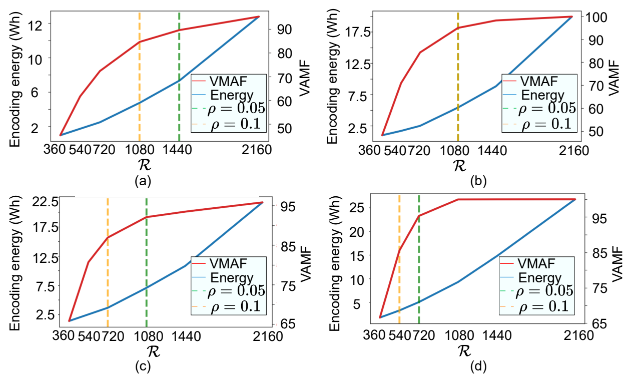


Figure 8: Impact of changing ρ values on video sequences: (a) sequence 72 ($E_Y = 73.07, h = 61.2, L_Y = 122.84; QP = 27$), (b) sequence 65 (41.61, 9.29, 90.10; 17), (c) sequence 23 (25.97, 28.37, 91.22; 22), (d) sequence 98 (37.39, 96.16, 109.35; 17).

7 Conclusions

This paper presents a novel method for predicting energy consumption in the context of selecting configurations for green encoding by employing low-pass anchors. The proposed method involves assessing the encoding and decoding durations of the anchor encodings that are created at reduced resolutions and utilizes lightweight machine learning models to anticipate the energy usage and video quality across all other representations within a given sequence. By analyzing these predictions, the encoding configuration is meticulously chosen to strike a balance between energy efficiency and acceptable quality reduction. Our evaluation, conducted on a dataset comprising 100 video sequences from the Inter4K dataset, illustrates that restricting the decrease in the VMAF score to 5 % results in substantial energy savings, specifically 51.22 % in encoding energy and 53.54 % in decoding energy, when compared to an encoding configuration selection that prioritizes maximum quality.

References

[1] AppLogic Networks, "The Global Internet Phenamena Report." https://www.applogicnetworks.com/phenomena, 2024. Retrieved: 2025-10-10.

[2] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann, "A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP," in *IEEE Communications Surveys Tutorials*, vol. 21, pp. 562–585, 2019.
[3] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, p. 133–144, 2011.
[4] Apple Inc., "HLS Authoring Specification for Apple Devices,"
[5] R. Farahani, A. Bentaleb, C. Timmerer, M. Shojafar, R. Prodan, and H. Hellwagner, "SARENA: SFC-Enabled Architecture for Adaptive Video Streaming Applications," in *ICC 2023-IEEE International Conference on Communications*, IEEE, 2023.
[6] R. Farahani, E. Çetinkaya, C. Timmerer, M. Shojafar, M. Ghanbari, and H. Hellwagner, "Alive: A Latency- and Cost-Aware Hybrid P2P-CDN Framework for Live Video Streaming," *IEEE Transactions on Network and Service Management*, 2023.
[7] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," in *IEEE Transactions on circuits and systems for video technology (TCSVT)*, vol. 22, pp. 1649–1668, IEEE, 2012.
[8] B. Bross, Y.-K. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan, and J.-R. Ohm, "Overview of the Versatile Video Coding (VVC) Standard and its Applications," *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, vol. 31, no. 10, pp. 3736–3764, 2021.
[9] A. Katsenou, J. Mao, and I. Mavromatis, "Energy-rate-quality Tradeoffs of State-of-the-art Video Codecs," in *Picture Coding Symposium (PCS)*, pp. 265–269, IEEE, 2022.
[10] T. Chachou, W. Hamidouche, S. A. Fezza, and G. Belalem, "Energy Consumption and Carbon Emissions of Modern Software Video Encoders," *IEEE Consumer Electronics Magazine*, pp. 1–16, 2023.
[11] R. Farahani, Z. Azimi, C. Timmerer, and R. Prodan, "Towards AI-assisted Sustainable Adaptive Video Streaming Systems: Tutorial and Survey," *arXiv preprint arXiv:2406.02302*, 2024.
[12] R. Farahani, H. Amirpour, F. Tashtarian, A. Bentaleb, C. Timmerer, H. Hellwagner, and R. Zimmermann, "RICHTER: Hybrid P2P-CDN Architecture for Low Latency Live Video Streaming," in *Proceedings of the 1st Mile-High Video Conference*, 2022.
[13] Z. Azimi Ourimi, R. Farahani, V. V. Menon, C. Timmerer, and R. Prodan, "Towards ML-Driven Video Encoding Parameter Selection for Quality and Energy Optimization," *Accepted in 16th International Conference on Quality of Multimedia Experience (QoMEX)*, 2024.
[14] A. Katsenou, V. V. Menon, A. Wieckowski, B. Bross, and D. Marpe, "Decoding Complexity-Rate-Quality Pareto-Front for Adaptive VVC Streaming," in *IEEE International Conference on Visual Communications and Image Processing (VCIP)*, pp. 1–5, 2024.
[15] A. V. Katsenou *et al.*, "Content-gnostic Bitrate Ladder Prediction for Adaptive Video Streaming," in *Picture Coding Symposium (PCS)*, IEEE, 2019.
[16] P. T. Rajendran, S. Afzal, V. V. Menon, and C. Timmerer, "Energy-Quality-Aware Variable Framerate Pareto-Front for Adaptive Video Streaming," in *IEEE International Conference on Visual Communications and Image Processing (VCIP)*, pp. 1–5, 2024.
[17] T. Huang *et al.*, "Deep Reinforced Bitrate Ladders for Adaptive Video Streaming," in *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2021.
[18] V. V. Menon, R. Farahani, P. T. Rajendran, M. Ghanbari, H. Hellwagner, and C. Timmerer, "Transcoding Quality Prediction for Adaptive Video Streaming," in *Proceedings of the 2nd Mile-High Video Conference*, 2023.
[19] Alliance For Telecommunications Industry Solutions, "Objective Video Quality Measurement Using A Peak-Signal-to-Noise Ratio Full Reference Technique," in *T1.TR.74-2001*, 2001.
[20] Z. Li, C. Bampis, J. Novak, A. Aaron, K. Swanson, A. Moorthy, and J. Cock, "VMAF: The Journey Continues," *Netflix Technology Blog*, vol. 25, no. 1, 2018.
[21] A. Stergiou and R. Poppe, "Adapool: Exponential Adaptive Pooling for Information-retaining Downsampling," *IEEE Transactions on Image Processing*, vol. 32, pp. 251–266, 2022.
[22] M. Ghasempour *et al.*, "Real-Time Quality-and Energy-Aware Bitrate Ladder Construction for Live Video Streaming," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2025.
[23] Y. O. Sharrab and N. J. Sarhan, "Aggregate Power Consumption Modeling of Live Video Streaming Systems," in *Proceedings of the 4th ACM Multimedia Systems Conference (MMSys)*, pp. 60–71, 2013.
[24] C. Herglotz, E. Walencik, and A. Kaup, "Estimating the HEVC Decoding Energy Using the Decoder Processing Time," in *IEEE International Symposium on Circuits and Systems*, pp. 513–516, IEEE, 2015.
[25] B. O. Turkkan, T. Dai, A. Raman, T. Kosar, C. Chen, M. F. Bulut, J. Zola, and D. Sow, "GreenABR: Energy-Aware Adaptive Bitrate Streaming with Deep Reinforcement Learning," in *Proceedings of the 13th ACM Multimedia Systems Conference (MMSys)*, pp. 150–163, 2022.
[26] R. Farahani, V. V. Menon, and C. Timmerer, "Machine Learning-Based Decoding Energy Modeling for VVC Streaming," in *IEEE International Conference on Image Processing (ICIP)*, pp. 2671–2676, 2025.
[27] P. Lebreton and K. Yamagishi, "Quitting Ratio-based Bitrate Ladder Selection Mechanism for Adaptive Bitrate Video Streaming," *IEEE Transactions on Multimedia (TMM)*, 2023.
[28] M. Bhat *et al.*, "Combining Video Quality Metrics to Select Perceptually Accurate Resolution in a Wide Quality Range: A Case Study," in *IEEE International Conference on Image Processing (ICIP)*, IEEE, 2021.
[29] Z. Azimi *et al.*, "Decoding Complexity-Aware Bitrate-Ladder Estimation for Adaptive VVC Streaming," in *32nd European Signal Processing Conference*, 2024.
[30] T. Kohonen, "Self-Organizing Maps," *Springer Science & Business Media*, 2012.
[31] V. V. Menon, C. Feldmann, K. Schoeffmann, M. Ghanbari, and C. Timmerer, "Green Video Complexity Analysis for Efficient Encoding in Adaptive Video Streaming," in *Proceedings of the First International Workshop on Green Multimedia Systems*, p. 16–18, 2023.
[32] A. Wieckowski, J. Brandenburg, T. Hinz, C. Bartnik, V. George, G. Hege, C. Helmrich, A. Henkel, C. Lehmann, C. Stoffers, *et al.*, "VVenC: An Open and Optimized VVC Encoder Implementation," in *IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pp. 1–2, IEEE, 2021.
[33] J. Brandenburg, A. Wieckowski, A. Henkel, B. Bross, and D. Marpe, "Pareto-optimized Coding Configurations for VVenC, a Fast and Efficient VVC Encoder," in *IEEE 23rd International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–6, IEEE, 2021.
[34] J. Boyce, K. Suehring, X. Li, and V. Seregin, *JVET-J1010: JVET Common Test Conditions and Software Reference Configurations.* 2018.
[35] A. Wieckowski, G. Hege, C. Bartnik, C. Lehmann, C. Stoffers, B. Bross, and D. Marpe, "Towards a Live Software Decoder Implementation for the Upcoming Versatile Video Coding (VVC) Codec," in *IEEE International Conference on Image Processing (ICIP)*, pp. 3124–3128, IEEE, 2020.
[36] BCG-GAMMA and MILA, "CodeCarbon." Retrieved: 2025-10-10.
[37] "Running Average Power Limit Energy Reporting." Retrieved: 2025-10-10.
[38] "Python bindings to the NVIDIA Management Library." Retrieved: 2025-10-10.
[39] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis.* John Wiley & Sons, 2021.
[40] G. C. McDonald, "Ridge Regression," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 1, pp. 93–100, 2009.
[41] L. Breiman, "Random Forests," in *Machine Learning*, vol. 45, 2001.
[42] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.
[43] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Advances in Neural Information Processing Systems*, 2017.
[44] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer Perceptron and Neural Networks," *WSEAS Trans. on Circuits and Systems*, 2009.
[45] "GridSearchCV." Retrieved: 2025-10-10.
[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine Learning in Python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[47] H. Amirpour, R. Schatz, and C. Timmerer, "Between Two and Six? Towards Correct Estimation of JND Step Sizes for VMAF-based Bitrate Laddering," in *14th International Conference on Quality of Multimedia Experience (QoMEX)*, pp. 1–4, IEEE, 2022.
[48] A. Kah, C. Friedrich, T. Rusert, C. Burgmair, W. Ruppel, and M. Narroschke, "Fundamental Relationships Between Subjective Quality, User Acceptance, and the VMAF Metric for a Quality-based Bit-rate Ladder Design for Over-the-top Video Streaming Services," in *Applications of Digital Image Processing XLIV*, vol. 11842, pp. 316–325, SPIE, 2021.
[49] J. Ozer, "Finding the Just Noticeable Difference with Netflix VMAF," *Streaming Learning Center*, 2017.

Toward Sustainability-Aware LLM Inference on Edge Clusters

Kolichala Rajashekar, Nafiseh Sharghivand, Radu Prodan
Department of Computer Science
University of Innsbruck, Austria

Reza Farahani
Institute of Information Technology
University of Klagenfurt, Austria

Abstract

Large language models (LLMs) require substantial computational resources, leading to significant carbon emissions and operational costs. Although training is energy-intensive, the long-term environmental burden arises from inference, amplified by the massive global query volume. Cloud-based inference offers scalability but suffers from latency and bandwidth constraints due to centralized processing and continuous data transfer. Edge clusters instead can mitigate these limitations by enabling localized execution, yet they face trade-offs between performance, energy efficiency, and device constraints. This short paper presents a sustainability-aware LLM inference for edge clusters comprising NVIDIA Jetson Orin NX (8GB) and Nvidia Ada 2000 (16GB) devices. It aims to balance inference latency and carbon footprint through carbon- and latency-aware routing strategies, guided by empirical benchmarking of energy consumption and execution time across diverse prompts and batch (i.e., group of prompts) configurations. We compared baseline greedy strategies to carbon-aware and latency-aware strategies in prompt routing to specific hardware based on benchmarking information. Experimental evaluation shows that a batch size of four prompts achieves a trade-off between throughput, energy efficiency, while larger batches risk GPU memory saturation.

Keywords

Sustainability, Large Language Models, LLM inference, Carbon Footprint, Edge Computing.

1 Introduction

Large language models (LLMs) require immense computational resources, driving both high carbon emissions and substantial operational costs. Although LLM training is notoriously energy-intensive, the inference phase, where models process user prompts into responses, poses a greater long-term sustainability challenge due to the massive, continuous global query volume [1, 13, 21]. Cloud servers provide vast computational and memory resources; however, processing all inference in the cloud requires extensive data transmission, which can degrade real-time performance under varying bandwidth availability [11, 22]. Moreover, inference for large-scale models such as GPT-4 can emit, on a daily basis, a carbon footprint comparable to a significant fraction of their one-time training emissions, intensifying sustainability concerns amid accelerating AI adoption [9, 13]. To address these challenges, modern computing architectures increasingly integrate centralized cloud resources with distributed edge devices, where edge instances handle latency-sensitive and lightweight tasks locally, reducing response time and associated emissions, while cloud systems manage compute-intensive workloads at scale [10, 12, 17].

This hybrid paradigm is particularly crucial for LLM inference, where prompt complexity varies widely. For example, simple factual questions may require only modest computational effort, while tasks involving multi-step reasoning or tool use demand substantially greater processing power. However, current LLM inference systems still rely on coarse-grained heuristics, e.g., routing all reasoning-heavy or mathematical prompts to high-capacity models, ignoring hardware heterogeneity, inter-device communication latency, or dynamic energy profiles [4, 8, 16]. These oversights lead to inefficient resource utilization, elevated carbon emissions, and degraded performance in edge deployments.

This short paper introduces sustainable LLM inference on edge clusters composed of NVIDIA Jetson Orin NX (8GB) and Nvidia Ada 2000 (16GB) as representative edge hardware. We propose carbon- and latency-aware strategies informed by extensive benchmarking across diverse LLM prompts from established prompt datasets [1]. Through empirical evaluation of energy consumption, carbon emissions, and end-to-end latency, our results show that these strategies reduce emissions by up to 35 % and improve execution speed by 2–3x compared to greedy baselines. Furthermore, we analyze the effect of batch size, i.e., the number of prompts processed in parallel during a single inference pass to amortize computational overhead, across configurations of 1, 4, and 8. Experimental results shows that batch size of 4 provides the balance between end-to-end latency and energy efficiency, while a batch size of 8 increases GPU utilization at the cost of higher latency. These findings highlight the importance of benchmarking-driven, sustainability-aware LLM inference to enable efficient and environmentally responsible deployments on edge server clusters.

2 Motivation Example

To illustrate the need for sustainability-aware LLM inference, we deployed two quantized Gemma models on an edge cluster: Gemma-3-1B-it-qat on an NVIDIA Jetson Orin NX (8GB) of GPU memory and Gemma-3-12B-it-qat on an NVIDIA Ada 2000 (16GB). For a cloud baseline, we used the Google Gemini 2.0 Flash API [20]. We used Ollama and evaluated four representative prompts (P1–P4), summarized in Table 1, which cover reasoning, generative writing, and factual lookup. We used a judge model (cloud) that rates expected reasoning depth and token footprint, calculating prompt complexity scores (CS); scores are normalized to [0,1], higher is harder.

Performance analysis: We compare the performance of deployed LLM models using four key metrics: *inference time* (IT), *time-to-first-token* (TTFT), *tokens-per-second* (TPS), and *time-per-output-token* (TPOT) [2, 15]. As shown in Fig. 1, the Gemma-3-12B model achieves the shortest TTFT but incurs higher IT and TPOT on longer prompts, whereas Gemma-3-1B provides a more balanced efficiency profile. The cloud-based Gemini 2.0 Flash API delivers superior IT

[1]https://huggingface.co/datasets

Table 1: Prompts used in evaluation (CS: complexity score [0–1] from a judge model; higher is harder).

ID	Prompt	CS
P1	A group of five friends (Alice, Bob, Carol, David, Emily) are trying to decide who will buy tickets for a concert, prepare snacks, drive, and pick up drinks. Alice hates driving. Bob can only pick up drinks if he's not preparing snacks. Carol loves concerts and wants to buy tickets. David can only drive if Emily prepares snacks. Emily will not pick up drinks. Each friend must take exactly one task, and each task must be assigned to exactly one friend. Assign the tasks to each friend and explain your logical deduction step by step.	0.47
P2	Write a short story, approximately 500 words, about a sentient, self-repairing antique grandfather clock that secretly orchestrates minor, benevolent 'time anomalies' in a quiet, forgotten library. Introduce a skeptical new librarian who slowly uncovers the clock's secret. The story must include: The clock's motivation for its actions. Three distinct 'time anomalies' are caused. A moment of direct, non-verbal communication between the clock and the librarian. A surprising twist where the librarian, instead of exposing the clock, aids its efforts for an unexpected reason.	0.39
P3	What is the boiling point of water at standard atmospheric pressure?	0.08
P4	Who painted the Mona Lisa?	0.07

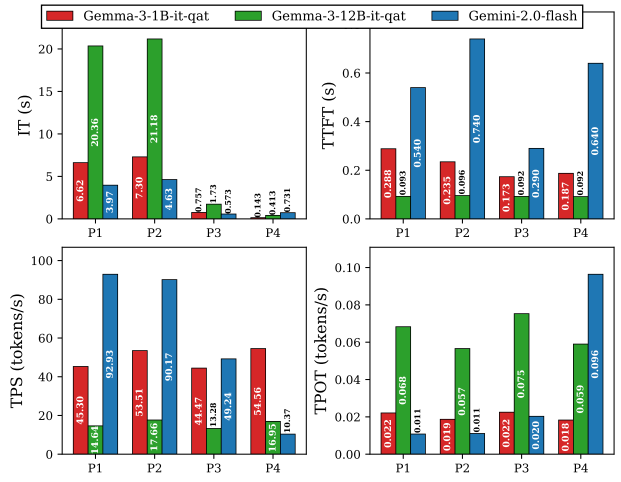


Figure 1: Comparison of inference performance metrics, IT, TTFT, TPS, and TPOT, across NVIDIA Jetson Orin NX (8 GB), Ada 2000 (16 GB), and the Gemini 2.0 Flash cloud API.

and TPS for complex prompts (P1, P2) but underperforms on simpler factual queries (P4), indicating bandwidth and dispatch overheads.

Sustainability analysis: Fig. 2 shows the measured carbon footprint (in CO_2eq) and power draw (in watts) obtained using the JetPack SDK [2] and the PyNVML library [3]. Gemma-3-1B emits roughly one-tenth the carbon of Gemma-3-12B on reasoning prompts (P1, P2), while both models exhibit low emissions on simpler ones (P3, P4). These results demonstrate that hardware-aware and model-adaptive inference can substantially reduce energy consumption and carbon emissions, underscoring the potential of sustainability-oriented deployment strategies for LLMs on edge clusters.

Key takeaway: Sustainability-aware LLM inference on edge clusters demands balancing performance, energy, and carbon efficiency.

[2]https://developer.nvidia.com/embedded/jetpack
[3]https://pypi.org/project/pynvml/

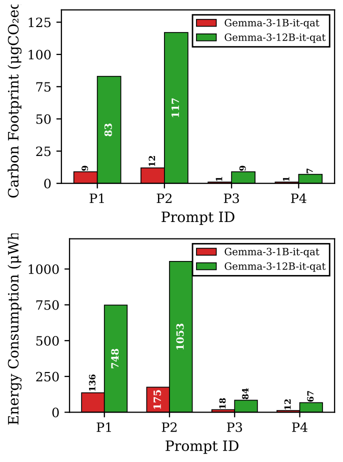


Figure 2: Carbon footprint and energy consumption for Gemma3-1B-it and Gemma3-12B-it models across prompts P1–P4.

Relying solely on either compact edge models or large cloud-based LLMs is suboptimal. Instead, hardware- and model-aware LLM inference is crucial for minimizing emissions while maintaining responsiveness. As shown in Fig. 1, lightweight models such as Gemma-3-1B achieve low latency for simple prompts, whereas Fig. 2 confirms substantial energy and carbon savings. For complex reasoning tasks, larger models, such as Gemma-3-12B or the Gemini 2.0 Flash API, offer superior output quality, underscoring the need for workload distribution across the edge–cloud continuum.

3 Benchmark Evaluation

We assessed our edge cluster, introduced in Section 2, using a mixed dataset that integrates prompts from multiple publicly available sources. The prompts spans diverse domains, including math reasoning (GSM8K) [7], extractive question answering (SQuAD) [19], dialogue summarization (DialogSum) [5], Python coding instructions [3], multiple-choice science reasoning (ARC-Challenge) [6], long-form summarization of arXiv papers, multi-turn dialogue continuation [18], and general long-form summarization [14]. From this composite benchmark of approximately 5000 prompts, we sampled 500 representative inputs to measure both end-to-end inference latency and the corresponding carbon footprint across the edge cluster. Table 2 summarizes the benchmark results, reporting average performance metrics across all evaluated configurations. These results provide an overview of the latency–energy trade-offs and serve as a practical guide for resource allocation in edge–cloud deployments. We implemented two LLM inference strategies:

(i) Carbon-aware: Assigns each prompt to the model with the measured lower carbon footprint, prioritizing emission reduction even if it increases latency.

(ii) Latency-aware: Employs a greedy heuristic that sorts prompts by decreasing average latency and assigns them to minimize total end-to-end execution time.

Baselines include assigning all prompts exclusively to either the Jetson Orin NX (8 GB) or the Ada 2000 (16 GB). The LLM inference results for batch sizes of 1, 4, and 8, summarized in Table 3, reveal

Table 2: Average inference metrics across edge devices and batch configurations.

Hardware	Batch Size	E2E Latency (s)	TTFT (s)	TPOT (s)	Token Count	Tokens/s (Throughput)	Energy (kWh)	Carbon (kgCO2e)
Ada 2000 16GB	1	3.39	0.26	0.03	69.62	20.54	6.35e-05	4.38e-06
	4	14.58	12.07	0.02	56.83	3.90	5.05e-05	3.49e-06
	8	26.82	24.00	0.03	63.97	2.39	5.73e-05	3.96e-06
Jetson 8GB	1	13.06	0.36	0.061	148	11.33	1.79e-05	1.23e-06
	4	15.08	1.13	0.063	149	9.88	4.89e-06	3.37e-07
	8	14.12	4.87	0.057	136	9.63	5.12e-06	3.53e-07

clear trade-offs between execution time and carbon footprint. For a batch size of 1, assigning all prompts to the NVIDIA Jetson (8GB) yields a total execution time of 1873.13 s with a carbon footprint of 0.000209 kg CO_2e, whereas using the NVIDIA Ada (16GB) reduces execution time to 1354.25 s but increases emissions to 0.000300 kg CO_2e. The *carbon-aware* strategy achieves the minimum footprint by directing roughly 85 % of prompts to the Jetson device, leveraging its energy efficiency for low-token tasks such as sentiment analysis. However, this causes high end-to-end (E2E) latency due to load imbalance from compute-intensive tasks such as Python coding. In contrast, the *latency-aware* strategy minimizes total E2E latency to 580.34 s by balancing workload distribution, assigning complex tasks to the Ada device, while maintaining a moderate carbon footprint of 0.000247 kg CO_2e. These results highlight the complementary roles of both devices: the Jotson device excels in lightweight workloads, whereas the Ada instance dominates in memory- and compute-intensive inference.

For a batch size of 4, the *Jetson-only* baseline achieves an execution time of 649.6 s with a carbon footprint of 0.000071 kg CO_2e, while the *Ada-only* configuration reduces execution time to 568.4 s but increases emissions to 0.000103 kg CO_2e. The *carbon-aware* strategy lowers emissions by approximately 33 % (0.000069 kg CO_2e) by routing around 80 % of prompts to the Ada device. However, E2E latency remains high due to memory constraints affecting compute-intensive tasks such as Python coding. In contrast, the *latency-aware* strategy achieves the shortest E2E latency, about twice as fast as the *Jetson-only* baseline, while maintaining a moderate carbon footprint of 0.000085 kg CO_2e, benefiting from balanced workload assignment that exploits the Ada device's efficiency for high-token prompts. Overall, a batch size of 4 provides a strong trade-off between throughput and energy efficiency, although minor accuracy degradation on the Ada device indicates limitations in handling larger model states.

At a batch size of 8, the *Jetson-only* baseline shows an execution time of 609 s with a carbon footprint of 0.000057 kg CO_2e, while the *Ada-only* setup reduces execution time to 533.6 s but increases emissions to 0.000084 kg CO_2e. The *carbon-aware* strategy yields the lowest footprint (0.000055 kg CO_2e) by routing approximately 75 % of prompts to the Jetson device. However, its E2E latency (552.4 s) remains elevated due to instability on high-token workloads. In contrast, the *latency-aware* strategy minimizes E2E latency to 266.8 s, roughly twice as fast as the *Jetson-only* baseline, while maintaining a moderate footprint of 0.000070 kg CO_2e, benefiting from the Ada device's greater stability in long-form summarization and other memory-intensive tasks. Although a batch size of 8 maximizes throughput, it introduces instability and accuracy degradation on the Jetson device, indicating that larger-memory configurations are preferable for high-batch inference workloads.

Table 3: Comparison of different LLM inference strategies across batch sizes 1, 4, and 8.

Strategy	Total E2E latency (s)	Total Carbon Footprint (kgCO2e)
	Batch Size 1	
All on Jetson (8GB)	1873.13	0.000209
All on Ada (16GB)	1354.25	0.000300
Carbon-Aware	1674.86	0.000204 (lowest)
Latency-Aware	580.34 (lowest)	0.000247
	Batch Size 4	
All on Jetson (8GB)	649.6	0.000071
All on Ada (16GB)	568.4	0.000103
Carbon-Aware	590.2	0.000069 (lowest)
Latency-Aware	284.2 (lowest)	0.000085
	Batch Size 8	
All on Jetson (8GB)	609.0	0.000057
All on Ada (16GB)	533.6	0.000084
Carbon-Aware	552.4	0.000055 (lowest)
Latency-Aware	266.8 (lowest)	0.000070

Cross-batch analysis reveals that overall latency decreases with larger batch size, as parallel token generation reduces TPOT. However, TTFT increases significantly, noticeably, limiting responsiveness in real-time applications. The carbon footprint per prompt declines with batching, since energy costs are amortized across multiple inputs. The Ada device consistently achieves higher accuracy, particularly at batch size 8, where the Jetson device exhibits errors due to memory saturation. Overall, batch size 4 offers the best trade-off between latency, carbon footprint, and accuracy, whereas batch size 8 maximizes throughput but demands at least 16GB of GPU memory for stable operation. The Jetson instance remains well-suited for lightweight, low-token tasks, while the Ada device excels in high-token, high-batch inference scenarios.

4 Conclusion and Future Work

This paper presented an empirical analysis of sustainability-aware LLM inference on heterogeneous edge server clusters. Our results show that increasing and balancing prompt batch sizes provides a clear trade-off between latency and carbon footprint, making it suitable for mixed workloads. In contrast, a larger batch size maximizes throughput but requires more memory for stability. In our experiments, the carbon-aware strategy reduces emissions by up to 35 % by leveraging the Jetson device efficiency for low-token tasks, while the latency-aware strategy achieves 2-3x faster execution times through balanced load distribution. These findings highlight the importance of dynamic, complexity-aware LLM inference for optimizing performance and minimizing environmental impact. Future work will investigate scalability for unseen prompts and adaptive edge-server selection to advance sustainable LLM inference.

Acknowledgment

This work received funding from the Horizon Europe research and innovation program (grant agreement 101189771, DataPACT) and the Austrian Research Promotion Agency (FFG, grant agreement 909989, AIM AT Stiftungsprofessur für Edge AI).

References

[1] Zoha Azimi, Reza Farahani, Christian Timmerer, and Radu Prodan. 2025. Towards an Energy-Efficient Video Processing Tool with LLMs. In *Proceedings of the 4th Mile-High Video Conference*. 89–90.

[2] Kayhan Behdin, Yun Dai, Ata Fatahibaarzi, Aman Gupta, Qingquan Song, Shao Tang, Hejian Sang, Gregory Dexter, Sirou Zhu, Siyu Zhu, et al. 2025. Efficient AI in Practice: Training and Deployment of Efficient LLMs for Industry Applications. *arXiv preprint arXiv:2502.14305* (2025).

[3] Tarun Bisht. 2021. python_code_instructions_18k: A Large-Scale Dataset of Python Code Instructions. Dataset on Hugging Face. https://huggingface.co/datasets/iamtarun/python_code_instructions_18k_alpaca.

[4] Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. 2024. Routerdc: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems* 37 (2024), 66305–66328.

[5] Yulong Chen, Yang Liu, Liang Chen, and Yue Zhang. 2021. DialogSum: A Real-Life Scenario Dialogue Summarization Dataset. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Association for Computational Linguistics, Online, 5062–5074. doi:10.18653/v1/2021.findings-acl.449

[6] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv:1803.05457v1* (2018).

[7] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168* (2021).

[8] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruhle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. 2024. Hybrid llm: Cost-efficient and quality-aware query routing. *arXiv preprint arXiv:2404.14618* (2024).

[9] Reza Farahani, Zoha Azimi, Christian Timmerer, and Radu Prodan. 2024. Towards AI-Assisted Sustainable Adaptive Video Streaming Systems: Tutorial and Survey. *arXiv preprint arXiv:2406.02302* (2024).

[10] Reza Farahani, Narges Mehran, Sashko Ristov, and Radu Prodan. 2024. Heftless: A Bi-objective Serverless Workflow Batch Orchestration on the Computing Continuum. In *2024 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 286–296.

[11] Reza Farahani and Radu Prodan. [n. d.]. Serverless Large Language Models: Edge vs. Cloud Deployment Trade-offs. In *Book of Abstracts*. 29.

[12] Reza Farahani and Radu Prodan. 2025. EnergyLess: An Energy-Aware Serverless Workflow Batch Orchestration on the Computing Continuum. In *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*. IEEE.

[13] Zhenxiao Fu, Fan Chen, Shan Zhou, Haitong Li, and Lei Jiang. 2024. LLMCO2: Advancing Accurate Carbon Footprint Prediction for LLM Inferences. *arXiv preprint arXiv:2410.02950* (2024).

[14] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching Machines to Read and Comprehend. In *Proceedings of NIPS*. Introduces the CNN/DailyMail summarization dataset.

[15] Kunal Jain, Anjaly Parayil, Ankur Mallick, Esha Choukse, Xiaoting Qin, Jue Zhang, Íñigo Goiri, Rujia Wang, Chetan Bansal, Victor Rühle, et al. 2024. Intelligent router for llm workloads: Improving performance through workload-aware scheduling. *arXiv preprint arXiv:2408.13510* (2024).

[16] Aly M Kassem, Bernhard Schölkopf, and Zhijing Jin. 2025. How Robust Are Router-LLMs? Analysis of the Fragility of LLM Routing Capabilities. *arXiv preprint arXiv:2504.07113* (2025).

[17] Jiaxing Li, Chi Xu, Lianchen Jia, Feng Wang, Cong Zhang, and Jiangchuan Liu. 2024. EACO-RAG: Towards Distributed Tiered LLM Deployment using Edge-Assisted and Collaborative RAG with Adaptive Knowledge Update. *arXiv preprint arXiv:2410.20299* (2024).

[18] Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. 2017. Dailydialog: A manually labelled multi-turn dialogue dataset. *arXiv preprint arXiv:1710.03957* (2017).

[19] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Jian Su, Kevin Duh, and Xavier Carreras (Eds.). Association for Computational Linguistics, Austin, Texas, 2383–2392. arXiv:1606.05250 [cs.CL] doi:10.18653/v1/D16-1264

[20] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786* (2025).

[21] Clovis Varangot-Reille, Christophe Bouvard, Antoine Gourru, Mathieu Ciancone, Marion Schaeffer, and François Jacquenet. 2025. Doing More with Less–Implementing Routing Strategies in Large Language Model-Based Systems: An Extended Survey. *arXiv preprint arXiv:2502.00409* (2025).

[22] Zheming Yang, Yuanhao Yang, Chang Zhao, Qi Guo, Wenkai He, and Wen Ji. 2024. Perllm: Personalized inference scheduling with edge-cloud collaboration for diverse llm services. *arXiv preprint arXiv:2405.14636* (2024).

Orchestrating Hierarchical Federated Learning Pipelines with the AIoTwin Middleware

Ivan Čilić
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
ivan.cilic@fer.unizg.hr

Ana Petra Jukić
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
ana-petra.jukic@fer.hr

Katarina Vuknić
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
katarina.vuknic@fer.unizg.hr

Ivana Podnar Žarko
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
ivana.podnar@fer.unizg.hr

Abstract

This tutorial introduces participants to the practical deployment of Hierarchical Federated Learning (HFL) across the Cloud-Edge-IoT (CEI) continuum. It explains the concept of Federated Learning and its extension to a hierarchical setup, as well as the orchestration aspects of setting up and running HFL pipelines in CEI environments. The tutorial includes a hands-on session which is divided into two parts. In the first part, participants will use the Flower framework to set up and execute standard and hierarchical Federated Learning (FL) tasks, providing hands-on insight into FL without orchestration. The second part focuses on the AIoTwin orchestration middleware, demonstrating how it enables seamless setup and monitoring of HFL pipelines across heterogeneous edge, fog, and cloud environments. The middleware automates the deployment of FL services (clients, aggregators) and handles infrastructure changes affecting the pipeline and global model performance, such as clients joining or leaving a running FL task. Participants are guided through deploying their own custom FL tasks on a real-world distributed cluster, while exploring the middleware's capabilities for orchestration, resource management, and hierarchical coordination. The tutorial is ideal for researchers and practitioners interested in deploying and running edge-to-cloud FL tasks at scale.

The tutorial was presented as part of the 15th International Conference on the Internet of Things (IoT 2025), held on November 18, 2025, in Vienna, Austria. Website: https://iot-conference.org/iot2025/workshops-tutorials/.

Keywords

edge computing, Federated Learning, orchestration, MLOps

1 Introduction and Motivation

IoT systems are inherently distributed: devices are spread across multiple locations and different administrative domains, often using edge computing devices in their vicinity. Therefore, IoT environments are well-suited for deploying Federated Learning (FL) [2] pipelines to create new machine learning (ML) models based on the generated IoT data, while preserving privacy by avoiding the transfer of raw and sensitive data to the cloud. FL pipelines enable model training in local networks in a distributed manner and close to IoT data sources by utilizing nearby edge devices for local training on *client nodes*. However, traditional FL pipelines, which rely on a single central server for aggregation of local models (*cloud aggregator*), face significant limitations when scaled across the CEI continuum: (1) *Communication overhead*: The centralized architecture with a single aggregator incurs high communication costs as each client must send model updates to the cloud aggregator; (2) *Data heterogeneity*: Statistical variation in data across clients leads to poor model convergence; and (3) *Dynamicity of CEI environments*: Traditional FL lacks native mechanisms to handle varying client resources, fluctuating bandwidth, and frequent node failures inherent to edge environments.

To overcome obstacles (1) and (2), **Hierarchical Federated Learning** (HFL) has emerged [1]. HFL introduces an intermediate layer of edge servers to aggregate local client updates before communicating with the global cloud aggregator. This multi-tier architecture can significantly improve the communication efficiency, making HFL pipelines ideal for CEI deployments.

To address obstacle (3) and complex setups of HFL pipelines, we propose the **AIoTwin Framework for Adaptive Orchestration of (H)FL Pipelines**, which supports dynamic adaptation to changes in the CEI continuum. The AIoTwin approach goes beyond static pipeline deployment and requires continuous runtime monitoring of both infrastructure health (e.g., node failures, network conditions) and ML performance of a running HFL pipeline (e.g., model accuracy, convergence rate). By integrating an orchestrator, the system can autonomously make informed decisions and coordinate reconfiguration actions—such as dynamic client-to-aggregator reassignment—to maintain high Quality of Service (QoS) and maximize model performance within defined constraints (e.g., communication cost).

The AIoTwin orchestration middleware supports HFL deployments at scale without requiring manual configuration and management of the computing infrastructure, i.e., edge and cloud nodes. The middleware, built on Kubernetes, automates the deployment of HFL services (clients and aggregators) across the available infrastructure and responds to infrastructure changes that affect FL performance. In other words, the middleware is adaptive and can reconfigure an HFL pipeline at runtime based on the triggers which are either infrastructure-related (e.g., node failure), or related to ML performance (e.g., increase in training loss).

2 AIoTwin HFL Orchestration Middleware: Architecture and Mechanisms

The AIoTwin middleware for adaptive orchestration of HFL pipelines extends the two main components of a general-purpose orchestrator such as Kubernetes: orchestrator and node. The main role of the orchestrator is to manage the FL pipeline, while nodes execute FL-specific services, clients and aggregators. The FL orchestrator extends the general-purpose orchestrator with two new components: *FL Controller*, designed to control the FL pipeline, and *FL Configuration*, responsible for determining the best-fit pipeline configuration for a given environment. Given the benefits and complexity of HFL, the orchestration is primarily designed to support HFL setups, but it also supports flat FL setups.

In a typical (H)FL setup, an ML engineer defines an FL task with the following inputs: (i) an initial ML model, (ii) training parameters, and (iii) the orchestration objective. The initial ML model serves as the starting point from which all clients begin their training during the initialization phase of the pipeline. The training parameters include batch size, learning rate, and similar settings. The orchestration objective can be defined on a case-by-case basis, for example, to either optimize the model's performance within a predefined cost budget or to minimize the cost while achieving a specific performance target, such as a target level of accuracy or loss. Clients need to prepare their training datasets, perform local training, and update their local models based on received aggregated model updates.

In particular, a typical HFL pipeline is executed through the following phases:

(1) *Initialization phase*: Clients prepare their local data for training, and the initial ML model is distributed by the global aggregator (GA) to all clusters and, in turn, to the clients.
(2) *Local training*: Clients train their local models for a predefined number of epochs.
(3) Local aggregation: After local training, clients send their model updates to their dedicated local aggregator (LA) for aggregation. The aggregated model is distributed back to the clients in the cluster.
(4) *Global aggregation*: After a predefined number of local aggregations, usually called local rounds, the edge-aggregated model is sent to the GA for global aggregation. Subsequently, the global model is distributed back to the LAs which forward it to the clients in their cluster. This marks the beginning of a new global round.
(5) *Model convergence and completion*: When the model has converged, or a certain threshold has been reached, the pipeline execution is completed.

3 Tutorial Objectives

This tutorial offers a comprehensive, two-part introduction to building and orchestrating scalable FL pipelines. Participants first learn the fundamentals of standard flat FL using a widely adopted FL framework (Flower[1]) and then use our extensions to setup hierarchical FL deployments exclusively with Flower. In the second part, they apply advanced techniques for adaptive orchestration using the open-source tools from the AIoTwin project, built on top of Kubernetes.

The tutorial is designed for researchers, students, and practitioners interested in distributed ML, edge computing, and service orchestration. Participants should be familiar with Python and have a basic understanding of ML concepts (e.g., neural networks, loss functions). Prior experience with PyTorch is beneficial. All required software (Flower, AIoTwin components) is open-source and provided in pre-configured virtual environments or containers set up by the tutorial organizers.

Learning Outcomes: Upon completion, participants are able to: (1) Set up a standard FL pipeline for model training using the Flower framework. (2) Understand and implement the key architecture of HFL (client, local aggregator, global aggregator). (3) Apply advanced methods for dynamic and adaptive orchestration of HFL pipelines.

4 Tutorial Structure

The tutorial is organized and structured into two parts and includes the following three tasks focusing on practical and hands-on lessons:

(1) **Task 1**: Standard Federated Learning Setup with Flower
(2) **Task 2**: Hierarchical Federated Learning with AIoTwin Flower Wrapper
(3) **Task 3**: Orchestrating HFL pipelines with AIoTwin Middleware

The first part relates to the usage of the Flower framework with flat and hierarchical pipelines[2]. The goal of **Task 1** is to establish a working baseline for distributed model training and understand the limitations of a flat FL architecture. Participants use the Python-based Flower framework (version 1.7.0) and the widely-used CIFAR-10 dataset. Participants execute the aggregator and multiple clients, observe the aggregation process (e.g., FedAvg), and track the global model's convergence rate and communication requirements.

In **Task 2**, we transition the flat FL setup into a multi-tier HFL architecture using the AIoTwin Extension of the Flower Framework (*fl-service*). The implementation of the FL service is available in the following repository: https://github.com/AIoTwin/fl-service. In this setup, participants configure three distinct roles: global aggregator, local aggregator and client. Participants execute a tiered deployment, observing how the local aggregation step reduces the communication load on the cloud server and also track the global model's convergence rate.

In the second part of the tutorial, we present the framework for adaptive orchestration of HFL pipelines and explain the need for pipeline reconfiguration during runtime. We also introduce our original reconfiguration validation algorithm (RVA) [3] which evaluates pipeline reconfiguration after a specific number of global rounds following a reconfiguration. The target is to maximize the ML model accuracy within a specified communication cost budget.

[1]Flower Labs GmbH, "Flower A Friendly Federated Learning Framework," https://flower.ai/
[2]Instructions for Task 1 and 2 are specified here: https://github.com/AIoTwin/fl-service/tree/tutorial.

Building on the HFL architecture deployed in Task 2, **Task 3** provides a hands-on exercise for adaptive HFL pipeline management using the AIoTwin Framework for Adaptive Orchestration of FL Pipelines (*fl-orchestrator*), published as open-source software in the following repository: https://github.com/AIoTwin/fl-framework. Participants explore how configuration files (e.g., YAML) define the list of nodes and their roles, and start an HFL pipeline within a predefined cluster. Participants are divided into teams, and each team uses eight pre-configured virtual machines for one HFL task/pipeline. They can also set up an experiment with dynamic changes of the infrastructure: a new client joining a running pipeline[3].

Furthermore, we demonstrate how orchestration logic can be implemented to optimize an objective (e.g., maximize model accuracy) while satisfying a constraint (e.g., communication cost budget), mirroring real-world operational requirements of CEI deployments. More details on how an orchestration objective can be tailored to maximize model performance within a predefined communication budget can be found in [3].

5 Conclusion

This tutorial provides a unique and highly relevant opportunity to gain practical experience with HFL and its critical orchestration challenges in the context of real-world deployments in the CEI continuum. Through hands-on experimentation with the established Flower framework and the novel, open-source AIoTwin orchestration middleware, participants can move beyond theoretical understanding to actively implement adaptive, resilient, and communication-efficient distributed ML solutions. The skills gained from configuring, deploying, and managing HFL pipelines are directly applicable to developing next-generation AIoT systems that operate reliably in heterogeneous and dynamic real-world environments while shaping the future of decentralized intelligence.

6 Acknowledgments

This work has been supported by the Horizon Europe WIDERA program under the grant agreement No. 101079214 (AIoTwin). Ana Petra Jukić was supported by the Croatian Science Foundation under project no. DOK-NPOO-2023-10-1182.

References

[1] Lumin Liu, Jun Zhang, S.H. Song, and Khaled B. Letaief. 2020. Client-Edge-Cloud Hierarchical Federated Learning. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 1–6. doi:10.1109/ICC40277.2020.9148862

[2] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 54)*, Aarti Singh and Jerry Zhu (Eds.). PMLR, 1273–1282. https://proceedings.mlr.press/v54/mcmahan17a.html

[3] Ivan Čilić, Anna Lackinger, Pantelis A. Frangoudis, Ivana Podnar Žarko, Alireza Furutanpey, Ilir Murturi, and Schahram Dustdar. 2025. Reactive Orchestration for Hierarchical Federated Learning Under a Communication Cost Budget. In *2025 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. 1–7. doi:10.1109/ICMLCN64995.2025.11140562

[3]Instructions for Task 3 are provided here: https://github.com/AIoTwin/fl-orchestrator/blob/iot-conf-tutorial/tutorial/README.md.