https://doi.org/10.31891/2219-9365-2026-87-20
UDC 004.8:004.75:004.42

ZAIATS Pavlo
Oles Honchar Dnipro National University
https://orcid.org/0009-0003-7263-8572
e-mail: paul.zaiats@gmail.com

REVIEW OF APPROACHES FOR REAL-TIME PERFORMANCE GUARANTEES IN EDGE MACHINE LEARNING SYSTEMS

This article presents a structured and comprehensive review of approaches aimed at ensuring real-time performance guarantees in edge machine learning systems. It highlights the ongoing shift of artificial intelligence workloads from centralized cloud infrastructures toward edge environments, where data is generated and processed locally. While this transition improves latency, bandwidth usage, and data privacy, it also introduces critical challenges due to the limited computational resources of edge devices.

The core contribution of the study is the development of a multi-layered taxonomy that organizes existing techniques into four key domains: model-level optimization, system-level scheduling, runtime adaptation, and hardware-software co-design. Each layer addresses different aspects of the real-time constraint problem. Model-level techniques focus on reducing computational complexity through methods such as quantization, pruning, and efficient architecture design. System-level strategies ensure timely execution through scheduling algorithms and resource allocation mechanisms. Runtime adaptation introduces dynamic adjustments to maintain performance under changing conditions, while hardware-software co-design aligns algorithms with the capabilities of underlying hardware.

The article emphasizes that no single technique is sufficient to guarantee real-time performance. Instead, it underlines the importance of integrating multiple approaches to balance trade-offs between accuracy, latency, and energy efficiency. This holistic perspective is particularly relevant for safety-critical applications, where missing execution deadlines can lead to system failure. Overall, the study provides a clear conceptual framework and valuable insights for designing robust, real-time edge AI systems.

Keywords: edge computing, machine learning, real-time, model compression, resource management, taxonomy.

ЗАЯЦЬ Павло
Дніпровський національний університет імені Олеся Гончара

ОГЛЯД ПІДХОДІВ ДО ЗАБЕЗПЕЧЕННЯ ГАРАНТІЙ ПРОДУКТИВНОСТІ В РЕЖИМІ РЕАЛЬНОГО ЧАСУ В СИСТЕМАХ МАШИННОГО НАВЧАННЯ НА ПЕРИФЕРІЇ

Ця стаття представляє структурований і комплексний огляд підходів, спрямованих на забезпечення гарантій реального часу в системах машинного навчання на периферії. У ній висвітлюється поступовий перехід навантажень штучного інтелекту від централізованих хмарних інфраструктур до периферійних середовищ, де дані генеруються та обробляються локально. Хоча такий перехід покращує затримки, використання пропускної здатності та конфіденційність даних, він також створює суттєві виклики через обмежені обчислювальні ресурси периферійних пристроїв.

Основним внеском дослідження є розробка багаторівневої таксономії, яка структурує існуючі підходи у чотири ключові домени: оптимізація на рівні моделі, планування на рівні системи, адаптація під час виконання та спільне проєктування апаратного і програмного забезпечення. Кожен із цих рівнів вирішує різні аспекти проблеми забезпечення обмежень реального часу. Методи на рівні моделі спрямовані на зменшення обчислювальної складності за допомогою таких підходів, як квантування, проріджування та ефективне проєктування архітектур. Системні стратегії забезпечують своєчасне виконання завдань завдяки алгоритмам планування та механізмам розподілу ресурсів. Адаптація під час виконання передбачає динамічне коригування параметрів для підтримання продуктивності за змінних умов, тоді як спільне проєктування апаратного і програмного забезпечення узгоджує алгоритми з можливостями базового обладнання.

У статті підкреслюється, що жоден окремий підхід не є достатнім для гарантування роботи в реальному часі. Натомість наголошується на необхідності інтеграції кількох підходів для досягнення балансу між точністю, затримкою та енергоефективністю. Такий цілісний підхід є особливо важливим для критично важливих систем, де порушення часових обмежень може призвести до відмови всієї системи. Загалом дослідження пропонує чітку концептуальну рамку та цінні практичні висновки для проєктування надійних систем машинного навчання на периферії з гарантіями реального часу.

Ключові слова: периферійні обчислення, машинне навчання, системи реального часу, стиснення моделей, управління ресурсами, таксономія.

Стаття надійшла до редакції / Received 10.06.2026
Прийнята до друку / Accepted 06.07.2026
Опубліковано / Published 10.09.2026

INTRODUCTION

Over the past decade, artificial intelligence workloads have drifted away from the cloud and closer to the devices that generate their data. This gradual shift (driven by the rapid spread of small networked cameras, sensors, and embedded boards in homes, factories, and cities) has changed how and where computation takes place. Handling at least part of this data directly on-device shortens communication paths, trims network traffic, and, as a side effect,

improves data confidentiality. Yet the convenience of local processing comes with a burden: edge devices usually have very limited CPU power, memory, and battery life.

At the same time, machine-learning models have grown deeper and heavier, creating a tension between accuracy and timeliness. Getting a high-quality prediction is not enough for real-time tasks since the result must also arrive within a strict time window. In such contexts, correctness has a temporal dimension – a slightly late answer can be as harmful as a wrong one. For instance, if the perception unit of an autonomous vehicle or the controller of an industrial robot misses just a single deadline, the entire system may fail rather than simply slow down [1].

To address these challenges, this study establishes a taxonomy of current approaches, organizing them into a layered framework designed to maintain predictable behavior in edge-AI systems operating under real-time constraints.

OBJECTIVES

The primary goal of this research is to provide a comprehensive analysis of methods for ensuring real-time performance in resource-constrained edge environments. Moving beyond isolated optimization techniques, this study aims to present a holistic view of the edge-ML stack through three objectives:

- Categorize literature: this study classifies current approaches into a layered framework consisting of four logical areas: model-level optimization, system-level scheduling, runtime adaptation, and hardware-software co-design.
- Evaluate trade-offs: the study analyzes the inherent tension between predictive accuracy, model size, and execution latency. It examines how aggressive compression impacts performance and investigates the balance between energy savings and transmission delays in offloading scenarios.
- Define a holistic integration strategy: the objective is to demonstrate that static design gains must be paired with dynamic system management to achieve robust real-time guarantees. The study highlights that the most resilient solutions require integrating these multiple layers to ensure firm deadlines are met.

TAXONOMY OF REAL-TIME EDGE ML APPROACHES

Model-level optimization and compression

Optimizing a neural network involves adjusting how numbers are represented within the model and how its layers exchange data. These adjustments must keep a balance between improving efficiency and maintaining accuracy. In practice, this means reducing memory use and eliminating redundant computations while preserving the model's essential structure. Common techniques for achieving this include quantization, pruning, and weight clustering.

Quantization is the most widely used method since it is the simplest one. It reduces the precision of weights and activations by replacing 32-bit floating-point values with 8-bit integers (sometimes even fewer bits). In this way, smaller representations reduce memory traffic and enable faster arithmetic on hardware optimized for integer operations. In simple post-training workflows, quantization is applied directly to an existing model, whereas in *quantization-aware training,* the reduced-precision effects are simulated during learning, allowing the network to adjust and retain more of its original accuracy [2, 3].

In contrast to quantization, **pruning** changes the network's structure itself. **Pruning** eliminates connections or neurons that contribute little to a model's output, leaving behind a lighter network that performs fewer calculations during inference.

Magnitude pruning is the simplest method. It drops the weights with the smallest absolute values. Another technique is called Taylor pruning, and it is more analytical. It goes further by estimating how strongly each weight affects the loss before deciding which ones to cut [4]. In practice, the choice between these two approaches reflects the trade-off a designer wants to make between accuracy and model size.

A related approach is weight clustering. It trims redundancy instead of deleting parameters outright. Here, similar weights are gathered into clusters and replaced by shared centroid values, sharply reducing the number of unique parameters that must be stored and, in turn, the memory required [3].

Despite different mechanics of quantization, pruning, and weight clustering, all these techniques rely on the same idea: a network's knowledge can often be represented more compactly without meaningfully changing what it predicts.

Another area of research investigates how to construct efficient networks and train them effectively from the start. Work in this area includes knowledge distillation, compound model scaling, and the development of lightweight convolutional modules.

Knowledge distillation uses a teacher-student model where a smaller "student" network learns to imitate the behavior of a larger and more precise "teacher". The student trains on labeled data and replicates the teacher's soft output distributions or intermediate feature maps. Through this transfer process, the smaller model preserves much of the original network's predictive power and uses significantly fewer parameters, resulting in lower runtime costs.

Distillation can be performed offline, using a fixed pre-trained teacher; online, where teacher and student are optimized jointly; or even in self-distillation mode, where a single model gradually teaches its own simplified version [2].

Compound model scaling approaches efficiency from a different angle. Instead of enlarging a network along just one axis (depth or width), it adjusts several dimensions together, including the input resolution. A single compound coefficient controls the scaling of all dimensions. Such an approach was introduced in EfficientNet, and it maintains a better balance between accuracy and computation across architectures [5].

Standard convolutions quite often use expensive calculations that are prohibitive for the edge. Some architectures introduced cheaper primitives in order to reduce the multiply-accumulate count. Architectures such as MobileNet and ShuffleNet became popular for their speed, driven by such innovations as depthwise separable convolutions, pointwise group convolutions, pointwise bottlenecks, and so on. By reducing the arithmetic load per layer, these designs made high-performance inference on constrained hardware possible [2, 3, 6].

Overall, these methods show that efficiency goes beyond compressing trained models. By embedding compact design principles directly into the network architecture, engineers can balance accuracy and speed from the very start of the design process.

Analysis and Trade-Offs

Compression and architectural redesign often shrink a model's size, but their effect on real-time performance can vary greatly across hardware and runtime environments. Speed-up depends on more than the degree of compression. Hardware accelerators, kernel efficiency, and data-transfer overhead all play a major role. In practice, an optimization that speeds up inference on one platform may slow it down on another.

A good example can be found in the quantization study [2]. Converting 32-bit floating-point weights to 8-bit integers can greatly reduce a network's size, but the real speed gain depends on how efficiently the hardware handles integer arithmetic. When kernels are poorly optimized, the extra work for clipping, rounding, and scaling can cancel out the expected benefit. In one test, post-training quantization of ResNet-50 increased latency from about 177 ms to 2348 ms (more than a 13-fold slowdown) while pruning the same model reduced latency to roughly 155 ms [2]. Pruning cuts the math directly, whereas quantization adds conversion overhead that strictly requires hardware support to be effective.

Another challenge is how to balance accuracy with efficiency. As compression grows more aggressive (whether by pushing sparsity to extremes or reducing precision to very few bits), the model usually begins to lose representational power and, with it, predictive accuracy. Empirical studies suggest that moderate pruning levels can be tolerated, but once sparsity exceeds roughly 80%, accuracy may drop by an order of magnitude [7]. **Knowledge distillation** helps to ease this trade-off by passing knowledge from a large teacher network to a smaller student. The smaller model, however, can only learn as much as its limited capacity allows, so the gain in compactness comes with a ceiling on attainable accuracy.

Design-time optimization produces lighter (and often faster) models. Still, real improvements appear only when these static gains are paired with runtime strategies that use available resources wisely. This interdependence bridges the gap between static model compression and dynamic system management – the focus of the next category of techniques devoted to scheduling and resource allocation.

System-Level Scheduling and Resource Management

While model optimization focuses on the "what" (the ML model as a workload), this category of approaches is concerned with the "how", "when", and "where" of execution. The main goal of system-level scheduling is to allocate time and resources so that real-time deadlines are met, even on limited hardware. Approaches in this area range from classical scheduling methods to newer techniques such as computation offloading and network slicing.

Every real-time system relies on a **scheduler**, the component that decides the order in which tasks run. Classic scheduling methods, such as **Rate-Monotonic** (which prioritizes more frequent tasks) and **Earliest-Deadline-First** (which runs the task with the nearest deadline), still form the mathematical basis for predictable timing [1]. Applying these rules to heterogeneous, accelerator-rich systems is not simple. Blocking and self-suspensions can break the assumption that tasks are independent, and this makes it difficult to identify the system's *critical instant* (the worst-case timing scenario) [8]. More recent work even employs machine-learning schedulers. Techniques using Long Short-Term Memory (LSTM) networks and reinforcement learning can dynamically adjust task priorities based on predictive analysis of system load and task characteristics, outperforming traditional heuristics in systems with variable workloads [9].

Once timing priorities are set, the next question is *where* each computation should execute. Offloading allows a device to delegate part (or sometimes all) of its workload to a nearby edge server. Full offloading transmits the entire task, while partial offloading splits execution between device and server, trading communication delay against local energy savings [10].

When many applications share the same infrastructure, *network slicing* offers a way to isolate workloads and guarantee predictable service. By allocating fixed portions of bandwidth, CPU, and memory to each "slice," operators can support diverse latency and throughput requirements without interference among tenants [10].

Analysis and Trade-Offs

System-level optimization always involves trade-offs. In offloading, engineers must balance energy savings against latency. Sending computation to a remote server can extend battery life, but it also introduces transmission delays that may cancel out those gains.

Resource allocation presents a different challenge. Centralized systems provide the best results on paper, but the cost of coordination and state collection makes them inefficient at scale. Decentralized approaches remove this bottleneck by letting devices decide for themselves. Game-theoretic research (using Potential and Stackelberg models) proves that such autonomous systems can self-organize into a stable state [10].

In summary, system-level scheduling and resource management act as the overall control layer that keeps real-time edge ML systems organized and predictable. However, workloads and environments rarely remain constant for long. They can shift in ways that are hard to anticipate. To stay reliable under these changing conditions, such systems increasingly rely on adaptive runtime mechanisms, which are explored in the next section.

Runtime Adaptation and Dynamic Policy Enforcement

This family of techniques introduces a **flexible, context-aware layer of optimization.** Unlike static configurations, this layer adapts to the system's current state. During operation, the system can adjust model parameters, resource limits, or scheduling policies to reflect changes in network quality, temperature, or battery level. The goal is to maintain accuracy and meet timing deadlines, even when conditions fluctuate unexpectedly.

Adaptive model selection removes the dependency on a single static network. The system keeps a repository of pre-trained candidates, which are used for substitution based on requirements. The control logic replaces the active network according to real-time feedback and instantly engages the encoder-inference pair that aligns with the current operational limits [11].

Dynamic model-approximation methods modify a network's structure or parameters in real time to meet changing performance needs. A key example is runtime-evolutionary compression, where the compression process itself becomes adaptive. Frameworks such as AdaSpring automate this by selecting and combining operators (like channel merging or low-rank factorization) while the system is running. These adjustments are guided by hardware-aware metrics and situational factors, such as available processing power and remaining battery life. In this way, the framework avoids costly retraining while aligning the model's complexity with the device's current state [12].

Approximation-aware scheduling takes an interesting approach by moving timing logic to the layer level. Instead of improving end-to-end performance for the whole model, frameworks like ApNet enforce sub-deadlines at each stage of execution. This creates a continuous runtime monitor. When any specific layer violates its assigned time slot, the control logic triggers a substitution and replaces the slow computation with a pre-calculated approximation. This granular intervention guarantees that the end-to-end deadline is satisfied, isolating the system from local processing jitter [13].

Dynamic system-parameter control refers to techniques that continuously tune low-level operating parameters to keep performance stable as conditions change. A common example is thermal-aware resource management. High-performance edge devices can generate enough heat to reduce both speed and reliability. To prevent overheating, frameworks such as RT-TRM dynamically adjust CPU voltage, frequency, and task periods as ambient temperature changes. These real-time adjustments keep the device within safe thermal limits while still meeting its timing requirements [14].

We can make hardware use the lowest feasible speed for a specific set of tasks. This technique is called static frequency scaling. It improves efficiency and temperature control. Frameworks like CycleSolo/CycleTandem perform offline analysis and pre-compute the exact frequencies required for the specific tasks. By applying these frequencies statically before execution, the system removes the computational penalty of dynamic voltage and frequency scaling and still guarantees the schedule [8].

Analysis and Trade-Offs

There is a price for flexibility. Dynamic model selection improves robustness, but the monitoring logic uses CPU cycles. The system has to continuously estimate latency and prediction confidence. This background processing occupies CPU cycles and introduces timing jitter. Such issues can be minimized by sampling, which should reduce such side effects, but on the other hand, sampling reduces efficiency. Anyway, the common conclusion also works here: we need to find a balance.

There is also a physical penalty. Frequent model switching leads to cache thrashing, while maintaining a deep pool of alternatives saturates the system memory. This proves that software cannot outrun the physics of the device. Flexibility is limited by hardware constraints, which act as the foundation for the analysis in the next section.

Hardware-Software Co-Design and Acceleration

Methods in this group seek real-time guarantees by aligning software optimizations directly with the strengths of the hardware beneath them. Instead of treating processors and accelerators as generic resources, these approaches

tailor algorithms to the underlying architecture. They take advantage of hardware features such as memory hierarchies and parallel execution units.

Today's edge systems depend increasingly on specialized chips to handle computationally heavy ML workloads that would otherwise overwhelm the main CPU. Components such as GPUs, TPUs, and NPUs excel at machine learning tasks. These accelerators offer major improvements in both speed and energy efficiency over general-purpose CPUs [15]. Even low-cost options, such as Intel's Neural Compute Stick, can turn a small board like a Raspberry Pi into a capable inference engine. These compact devices enable complex models to run locally and in real time [15].

Neural Architecture Search (NAS) is a technique for automating the design of neural networks. And there is a Platform-aware NAS that goes further and treats hardware limits as design constraints. It integrates physical factors like latency and energy into the search algorithm. This moves the focus away from metrics like FLOPs. Instead, the system optimizes for the target device, discovering architectures that align with the hardware's specific capabilities. In the end, we get a model that has maximum speed and efficiency on a specific device.

A complementary line of work focuses on **efficient neural-network kernels** – the low-level computation libraries that execute the model's operations. One example is CMSIS-NN, a library of optimized kernels for Arm Cortex-M processors. Each operation is tuned to the chip's instruction set and memory layout, maximizing performance while keeping memory use low [2].

Analysis and Trade-Offs

We rely on pipelining and multi-level caches for throughput. But such architectural features introduce a penalty in the form of jitter. This latency instability complicates worst-case execution time (WCET) analysis and makes time bounding nearly impossible. For real-time systems that use classical models like Rate-Monotonic, this means that a mathematical safety guarantee is invalid because input variables cannot be trusted [1].

A key question in edge computing is whether to rely on general-purpose CPUs or to incorporate specialized accelerators. Accelerators deliver higher throughput and energy efficiency for targeted machine-learning workloads, while CPUs remain essential for managing a broader range of tasks. The right balance depends on the application's priorities (speed, cost, or flexibility), and in practice, these factors rarely align perfectly.

Understanding hardware-level metrics in hardware-software co-design is just as important as tuning the model itself. One example is arithmetic intensity – the ratio of computation to memory access. It provides a clearer view of how efficiently a device processes data. Metrics that reflect platform characteristics generally predict real-world performance more accurately than model-only measures that ignore hardware constraints. This is because arithmetic intensity directly relates to data movement, which is a primary driver of energy consumption on physical hardware [12].

Hardware-software co-design forms the foundational layer on which all other software-based strategies depend. It emphasizes that a robust real-time system requires a holistic design approach.

CONCLUSION AND FUTURE DIRECTIONS

Providing real-time performance guarantees for machine learning at the edge is a multidisciplinary challenge. There is no silver-bullet solution for it. This study outlines four complementary directions for achieving real-time performance in edge-based machine learning. The first focuses on model optimization and compression to produce efficient static networks. The second addresses system-level scheduling and resource management. The third introduces adaptive runtime control, allowing systems to respond dynamically to changing conditions. Finally, the fourth brings software and hardware together through a co-design approach that aligns both sides of the system.

The literature as a whole points to one clear pattern: high, predictable performance rarely comes from a single technique. In practice, the most resilient solutions combine methods from several levels. A modern autonomous-driving stack, for instance, might use a pruned and quantization-aware model on an Edge-TPU platform, coordinated by an Earliest-Deadline-First scheduler and adjusted by a thermal-aware controller such as RT-TRM. All these elements help real-time systems keep their deadlines firm while preventing thermal overload.

Looking ahead, **edge hardware** will continue to grow more capable. At the same time, **machine-learning models** will keep expanding in size. Future progress will depend on how effectively these ideas can be brought together. The tighter the integration between compression, scheduling, adaptation, and hardware-aware design, the closer we move toward real-time intelligence at the edge.

References

1. Laplante, P. A., Ovaska, S. J. Real-Time systems design and analysis: Tools for the Practitioner. John Wiley and Sons. 2011. 584 p.
2. Lee, H., Lee, N., Lee, S. A method of deep learning model optimization for image classification on edge device. Sensors. 2022. 22(19), 7344. DOI: https://doi.org/10.3390/s22197344
3. Anirudh, S. Optimal compression of convolutional neural networks for edge devices. EEScholars. 2022. URL: https://eescholars.iitm.ac.in/sites/default/files/eethesis/ee18b073.pdf
4. Mou, A., Milanova, M. Performance Analysis of Deep Learning Model-Compression Techniques for audio classification on edge Devices. Sci. 2024. 6(2), 21. DOI: https://doi.org/10.3390/sci6020021

5. Tan, M., Le, Q., V. EfficientNet: Rethinking model scaling for convolutional neural networks. arXiv (Cornell University). 2019. DOI: https://doi.org/10.48550/arxiv.1905.11946
6. Zhang, X., Zhou, X., Lin, M., & Sun, J. ShuffleNet: an extremely efficient convolutional neural network for mobile devices. arXiv (Cornell University). 2017. DOI: https://doi.org/10.48550/arxiv.1707.01083
7. Chanda, D. G. Benchmarking of energy and latency of CNN models on Raspberry Pi and Energy-Aware weight based pruning of CNN model. 2024. URL: https://hdl.handle.net/20.500.12588/6965
8. D'souza, S., Rajkumar, R. CycleTandem: Energy-Saving Scheduling for Real-Time Systems with Hardware Accelerators. In IEEE Real-Time Systems Symposium (RTSS). 2018. (pp. 94–106). DOI: https://doi.org/10.1109/rtss.2018.00019
9. Gracias, A., Broklyn, P. LEVERAGING AI AND ML FOR PREDICTIVE SCHEDULING IN REAL-TIME OPERATING SYSTEMS. ResearchGate. 2025. URL: https://www.researchgate.net/publication/387958272_LEVERAGING_AI_AND_ML_FOR_PREDICTIVE_SCHEDULING_IN_REAL-TIME_OPERATING_SYSTEMS
10. Zhang, X., & Debroy, S. Resource Management in Mobile Edge Computing: A Comprehensive survey. ACM Computing Surveys. 2023. 55(13s), 1–37. DOI: https://doi.org/10.1145/3589639
11. Kalør, A. E., Ohtsuki, T. Black-Box Edge AI Model Selection with Conformal Latency and Accuracy Guarantees. arXiv (Cornell University). 2025. DOI: https://doi.org/10.48550/arxiv.2506.11391
12. Liu, S., Guo, B., Ma, K., Yu, Z., Du, J. AdaSpring: Context-adaptive and Runtime-evolutionary Deep Model Compression for Mobile Applications. Proceedings of the ACM on Interactive Mobile Wearable and Ubiquitous Technologies. 2021. 5(1), 1–22. DOI: https://doi.org/10.1145/3448125
13. Bateni, S., Liu, C. ApNet: Approximation-Aware Real-Time Neural Network. In Real-Time Systems Symposium (RTSS). 2018. (pp. 67–79). DOI: https://doi.org/10.1109/rtss.2018.00017
14. Lee, Y., Chwa, H. S., Shin, K. G., Wang, S. Thermal-Aware resource management for embedded Real-Time systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2018. 37(11), 2857–2868. DOI: https://doi.org/10.1109/tcad.2018.2857279
15. Rao, D. S., Adinarayana, S., Samantray, O. P., Rao, I. S. S. EDGE AI: MERGING IOT AND MACHINE LEARNING FOR REAL-TIME ANALYTICS. Xoffencer International Book Publication house. 2024. 258 p.