http://doi.org/10.35784/iapgos.8809 *received: 25.11.2025 | revised: 12.12.2025 | accepted: 17.12.2025 | available online: 20.12.2025*

# IMPACT OF CUSTOMIZABLE ORCHESTRATOR SCHEDULING ON MACHINE LEARNING EFFICIENCY IN EDGE ENVIRONMENTS

**Konrad Cłapa, Krzysztof Grudzień, Artur Sierszeń**
Technical University of Lodz, Institute of Applied Computer Science, Lodz, Poland

***Abstract.*** *This article explores the impact of custom scheduling strategies on the performance of machine learning workflows at the edge by using the case of Kubernetes scheduling. Optimizing machine learning (ML) workloads on resource-constrained edge devices has become a significant scientific challenge addressed by multiple studies. The severe limitations of edge systems in processing power, memory, and energy render conventional cloud-native schedulers ineffective, leading to poor resource utilization and degraded performance. While numerous advanced, data-driven solutions have been proposed for large-scale systems, their complexity and overhead are often impractical for edge deployments. In contrast, this work investigates a simpler, lightweight scheduling mechanism for CPU-based workloads that provides efficient and predictable performance without relying on historical data, making it well-suited for the unique requirements of the edge.Using a lightweight K3s cluster integrated with Kubeflow Pipelines, we investigate how varying binpacking functions influence resource allocation and training efficiency of a CNN model on the MNIST dataset. Our experiments demonstrate that tailored scheduling configurations can lead to noticeable improvements in training times and hardware utilization across different edge cluster sizes. The results offer actionable insights for optimizing AI workloads in resource-constrained edge environments.*

**Keywords**: machine learning, artificial intelligence, cloud computing, edge computing, Internet of Things

# WPŁYW DOSTOSOWYWALNEGO HARMONOGRAMOWANIA ORKIESTRATORA NA WYDAJNOŚĆ UCZENIA MASZYNOWEGO W URZĄDZENIACH BRZEGOWYCH

***Streszczenie.*** *Ten artykuł analizuje wpływ niestandardowych strategii harmonogramowania na wydajność przepływów pracy uczenia maszynowego na brzegu (edge), na przykładzie harmonogramowania w Kubernetes. Optymalizacja obciążeń uczenia maszynowego (ML) na urządzeniach brzegowych o ograniczonych zasobach stała się istotnym wyzwaniem naukowym, które jest przedmiotem wielu badań. Poważne ograniczenia systemów brzegowych w zakresie mocy obliczeniowej, pamięci i energii sprawiają, że tradycyjne, chmurowe schedulery są nieskuteczne, co prowadzi do słabego wykorzystania zasobów i spadku wydajności. Podczas gdy liczne zaawansowane, oparte na danych rozwiązania zostały zaproponowane dla systemów na dużą skalę, ich złożoność i narzut często czynią je niepraktycznymi w środowiskach brzegowych. W przeciwieństwie do tego, niniejsza praca bada prostszy, lekki mechanizm harmonogramowania dla obciążeń opartych na CPU, który zapewnia efektywną i przewidywalną wydajność bez potrzeby korzystania z danych historycznych, dzięki czemu dobrze odpowiada unikalnym wymaganiom środowisk edge. Wykorzystując lekki klaster K3s zintegrowany z Kubeflow Pipelines, badamy, w jaki sposób różne funkcje „upakowywania zadań" wpływają na alokację zasobów i efektywność treningu modelu CNN na zbiorze danych MNIST. Nasze eksperymenty pokazują, że odpowiednio dostosowane konfiguracje harmonogramowania mogą prowadzić do zauważalnych usprawnień w czasie treningu i wykorzystaniu sprzętu w różnych rozmiarach klastrów brzegowych. Wyniki dostarczają praktycznych wskazówek dotyczących optymalizacji obciążeń AI w środowiskach edge o ograniczonych zasobach.*

**Słowa kluczowe**: uczenie maszynowe, sztuczna inteligencja, przetwarzanie w chmurze, przetwarzanie brzegowe, Internet rzeczy

## Introduction

The widespread adoption of artificial intelligence (AI) applications has intensified the demand for efficient and localized machine learning (ML) model training directly on edge computing devices [12, 16]. In contrast to cloud-native environments that benefit from virtually unlimited computational power, abundant energy, and large-scale infrastructure, edge systems operate under strict resource limitations – including constrained processing capabilities, reduced memory, limited energy availability, and compact physical design [12, 13]. These constraints substantially increase the complexity of orchestrating and optimizing compute-intensive ML workloads in edge environments.

Performing data processing locally at the edge is becoming increasingly important to ensure compliance with data sovereignty requirements and to minimize dependency on network connectivity. This approach not only reduces data transmission latency but also enables real-time decision-making for latency-sensitive applications [12, 14]. Typical ML scenarios at the edge include real-time inference, continual learning, on-device fine-tuning to adapt to local contexts, hyperparameter optimization, and dynamic model updates to address environmental or sensor variability [2, 12]. Such adaptive capabilities are particularly valuable in domains such as autonomous systems, industrial IoT, remote sensing, and personalized healthcare, where localized learning can significantly enhance reliability and performance [14, 16].

Although cloud-native frameworks - such as Kubernetes and Kubeflow – provide robust orchestration mechanisms for managing ML workflows in distributed systems [7, 13], their default schedulers are primarily designed for resource-abundant data centers. Consequently, these mechanisms often lead to inefficient resource utilization, extended training durations, and suboptimal performance when applied to the inherently constrained edge environments [14].

Overcoming these challenges requires a more profound understanding of specialized scheduling strategies tailored to the operational constraints and optimization priorities of edge computing, with the aim of improving the efficiency and performance of ML model training in such contexts [4, 13].

## 1. Scope of research

This study seeks to systematically analyze and quantify the impact of customized Kubernetes scheduler characteristics, particularly those employing the bin-packing strategy [7], on the efficiency and performance of machine learning (ML) model training within a multi-node edge computing environment. By applying various bin-packing functions, the research controls and evaluates how differences in scheduling behavior influence workload placement relative to CPU resource availability. The investigation focuses on assessing how these scheduling variations affect the training efficiency of a Convolutional Neural Network (CNN) [9], a widely adopted deep learning architecture, using the benchmark MNIST handwritten digit classification dataset [10]. The overarching goal is to provide empirical insights into the effectiveness of different bin-packing configurations for resource management in ML training workflows, thereby advancing the optimization of decentralized edge AI systems.

## 2. Related work

In recent years, industry has increasingly deployed large-scale AI clusters composed of thousands of CPUs and GPUs to train deep learning models using frameworks such as MXNet [3], TensorFlow [1], and Petuum [15]. Because deep learning training is resource-intensive and time-consuming, efficient

scheduling has become both a critical operational requirement and an important scientific research topic aimed at improving resource utilization, scalability, and performance in distributed environments.

Early studies extended general-purpose schedulers to support distributed learning workloads, but these systems relied on static resource allocation, limiting adaptability under varying load conditions. Later research introduced dynamic and learning-based schedulers that optimize resource allocation in real time. Among these, the Deep Reinforcement learning enhanced Scheduler (DRS) models scheduling as a Markov Decision Process, improving utilization and load balance compared to the default Kubernetes scheduler [6].

Other efforts concentrate on modular, plugin-based architectures such as Volcano, which augments Kubernetes with batch and AI workload support through pluggable policies like gang scheduling, fair share, and topology awareness. The Volcano system has been analyzed in simulation environments such as K8sSim, which integrates multiple Volcano scheduling algorithms to evaluate their performance on real cluster traces [5]. In related work, Liu and Guitart [11] employ Volcano as a baseline for containerized high-performance computing (HPC) workloads and propose fine-grained scheduling strategies that decompose jobs across hierarchical levels to improve response time and makespan.

In contrast to these complex and data-driven solutions, this work investigates a simpler scheduling mechanism that does not depend on large datasets or historical performance information. The proposed approach targets CPU-based workloads and aims to maintain a small computational footprint while delivering efficient and predictable scheduling decisions. This design emphasizes lightweight implementation, transparency, and low operational overhead, making it well-suited for edge deployments and low-cost environments where simplicity and resource efficiency are essential.

## 3. Laboratory setup

To support efficient and reproducible experimentation, the laboratory setup was structured into two core layers: hardware and software. At the initial phase deliberate choice was made to prioritize system stability and flexibility, resulting in a fully virtualized test environment hosted on a Linux workstation. This approach facilitated the construction and teardown of test scenarios with ease, while avoiding hardware-related inconsistencies that could skew results. Each node in the Kubernetes cluster was virtualized and configured using the Ubuntu Linux distribution, allowing the entire environment to be codified and rapidly rebuilt as needed.

As part of the environment preparation phase, various Kubernetes distributions were evaluated for their suitability in edge deployment scenarios. Two candidates were initially tested, with K3s and MicroK8s emerging as viable options based on their performance and resilience under adverse conditions such as suspensions, unexpected shutdowns, and power losses. These conditions are especially relevant to edge servers, particularly in air-gapped or remote environments, where physical access is limited and failure recovery must be autonomous.

In our experiments, both K3s and MicroK8s were evaluated as lightweight Kubernetes distributions tailored for edge deployment. MicroK8s offers a comprehensive set of built-in features and ready-to-use application blueprints, which makes it attractive for rapid prototyping and simplified setup. However, during testing, K3s demonstrated superior operational stability, especially under adverse conditions.

K3s was more resilient to unexpected system failures, such as power losses or abrupt hardware reboots. It maintained cluster integrity and resumed workloads reliably across multiple unplanned interruptions. This robustness makes K3s particularly well-suited for edge environments, where systems may be exposed to unstable power sources or severe environmental conditions.

On top of these Kubernetes distributions, we deployed Kubeflow, an open-source platform for managing machine learning workflows in containerized environments. Kubeflow provided a complete pipeline orchestration layer, allowing us to automate model training and evaluation tasks while tracking execution performance with high granularity.

The combination of K3s and Kubeflow ultimately offered a lightweight yet resilient solution for edge ML deployments, ensuring both efficient resource utilization and operational continuity in constrained and unpredictable environments.

Following the initial experiments conducted in a virtualized environment , the research progressed to a more realistic edge-like deployment using physical hardware. The updated experimental setup consists of three Kubernetes clusters, each comprising between one and three K3s nodes, deployed on Intel NUC 11 Enthusiast Kit mini-PCs.

Each Intel NUC is equipped with the following specifications:

- Processor: Intel® Core™ i7-1165G7
- Architecture: 4 physical cores / 8 threads
- Base frequency: 2.80 GHz, with Turbo Boost up to 4.70 GHz
- Memory support: Up to 64 GB DDR4 RAM
- GPU: NVIDIA® GeForce RTX 2060 (discrete GPU) – not used in this study as all computations were CPU-based

This hardware was selected for its compact form factor, low power consumption, and sufficient processing capability to simulate realistic edge deployments under resource constraints. Despite the availability of a discrete GPU, all experiments were executed using only CPU resources, in alignment with the project's goal of targeting low-footprint, energy-efficient ML processing on the edge.

Each NUC hosted a single K3s node, serving as both the control plane and the worker node. This mirrors common edge scenarios where dedicated infrastructure is not available, and nodes must be self-sufficient.

Kubeflow Pipelines were deployed across these clusters, orchestrated by the Kubernetes scheduler. Component distribution was left to the default scheduling logic – no affinity or anti-affinity rules were applied – allowing workloads to be assigned flexibly based on available CPU resources and binpacking policies under evaluation.

## 4. Methodology

To evaluate the impact of Kubernetes scheduling strategies on edge machine learning performance [4], we designed an experiment centered on the MNIST dataset [10], using a Convolutional Neural Network (CNN) model. The entire machine learning workflow – data preprocessing, training, and evaluation – was implemented as a Kubeflow pipeline, with each stage deployed in a separate container. This modular approach provided visibility into the resource usage of each stage and facilitated isolated execution control.

### 4.1. Experiment design

To establish a performance baseline, we deployed the pipeline on three K3s cluster configurations:

- Cluster A: Single-node,
- Cluster B: Two-nodes,
- Cluster C: Three-nodes.

These configurations allowed us to analyze how cluster size influences execution behavior using the default Kubernetes scheduler [4]. After the baseline was established, all further experiments were conducted exclusively on Cluster C, leveraging its multi-node architecture to study parallel execution dynamics.

Our core research objective was to assess whether the configuration of the request-to-capacity ratio curve in the scheduler

scoring function has any measurable effect on pipeline performance. To evaluate this, we developed a custom binpacking-based scheduler logic. By adjusting how CPU request loads influenced node scoring, we were able to test whether workloads should be packed more densely or spread more evenly across nodes.

Five distinct binpacking scoring functions were implemented and applied in separate experimental runs. These configurations allowed us to explore the impact of scheduling characteristics on performance metrics under increasing system load.

To systematically test performance under varying conditions, the pipeline was executed with increasing levels of parallelism, starting from a single pipeline and scaling up to $N = 30$ concurrent executions. For each configuration, we recorded four key execution time intervals: pipeline creation time ($t_c$), execution time of ML workload ($t_e$), retrieval time for final results ($t_r$), deletion time for the completed pipeline ($t_d$), The total pipeline experiment time is the sum of the above times as depicted in Fig 1.

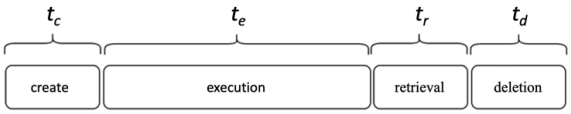



*Fig. 1. Time intervals*

Of these recorded intervals, the execution time of the ML workload was the dominant component, exceeding the other metrics by at least two orders of magnitude, thereby rendering the creation, retrieval, and deletion times negligible and establishing the workload execution as the sole focus of our performance analysis.

Each test scenario was repeated at least five times to ensure statistical reliability. Outlier values were removed using a dispersion-based filtering method, and for each case scenario minimum, maximum, and average times were computed. This methodology enabled a detailed analysis of two key aspects: the relationship between total execution time and the number of concurrent pipeline executions, and the effect of different scheduler scoring functions on performance behavior.

This structured experiment allowed us to draw meaningful conclusions about how Kubernetes scheduling logic can influence the efficiency and scalability of CNN-based ML workloads on CPU-constrained edge infrastructure.

## 4.2. Experimental setup

The architecture of the experimental environment is presented in Fig. 2, which illustrates the main components and their interconnections between the Measurement System and the Edge System.

The Edge System hosted the actual execution environment. It consisted of a lightweight K3s cluster running Kubeflow, Argo Workflows, and MinIO for workflow orchestration, experiment management, and data storage, respectively. The K3s Orchestrator coordinated a set of nodes (Node 1 to Node N) operating on a common Virtualization Layer and underlying Hardware infrastructure. Communication between the Measurement and Edge Systems occurred via the Kubeflow API, ensuring seamless remote control and data collection.

The Measurement System was responsible for defining and controlling the benchmarking process. It included the Kubeflow Pipelines definitions, a Benchmarking System that automated workload execution and data collection, and a Results module for storing and processing experiment outputs. To support automated testing, a custom benchmarking script was developed using existing open-source libraries. This script interacted with the Kubeflow API to programmatically create and execute multiple ML pipelines, simulating varied operational loads.

Crucially, the execution time of each container was measured internally by the Kubeflow system itself. By relying on internal instrumentation rather than external tools, the setup avoided any latency introduced by network communication or data preprocessing overhead. This allowed for precise measurement of the actual computation time for each pipeline stage.

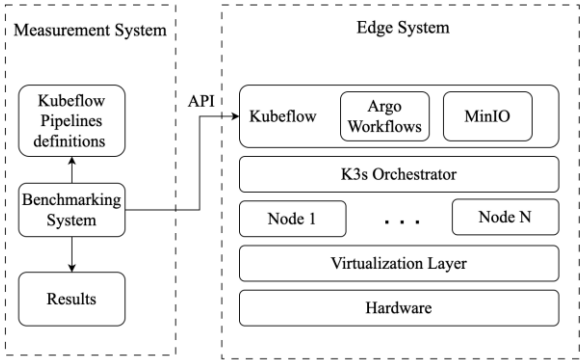



*Fig. 2. Experimental setup*

All timing and performance data were systematically logged to files for further processing and analysis. To ensure measurement integrity, the hardware used for the experiment was dedicated exclusively to testing and remained free from unrelated workloads throughout the entire period. This isolation helped prevent external factors from influencing execution results, thereby improving the repeatability and reliability of the findings.

In the experiment we have applied custom scheduling characteristics however the mechanism for scheduling remained as per default scheduler architecture. For better understanding of the experiment setup it is important to understand the different phases of workload scheduling which are as follows.

The Kubernetes scheduler works in two main phases:

1) Filtering (Predicates): In this phase, the scheduler identifies a list of feasible nodes by ruling out any node that cannot meet a pod's basic requirements. This includes checks like:
- PodFitsResources: Does the node have enough free CPU and memory to satisfy the pod's requests?
- NodeAffinity: Does the node match the pod's affinity rules?
- Taints and Tolerations: Does the pod tolerate the node's taints?
2) If no feasible nodes are found, the pod remains in a Pending state.
3) Scoring (Priorities): Once a list of feasible nodes is identified, the scheduler assigns a numerical score to each one to find the "best" fit. The default scoring strategy, called NodeResourcesLeastAllocated, favors nodes that are least allocated in terms of CPU and memory. This is a "spread" strategy, which aims to distribute pods across as many nodes as possible to prevent any single node from becoming a single point of failure and to make room for larger pods in the future. The node with the highest score is chosen for the pod.

Figures 3 through 7 illustrate the scheduler scoring characteristics corresponding to the five evaluated scheduling configurations used in the experiments.

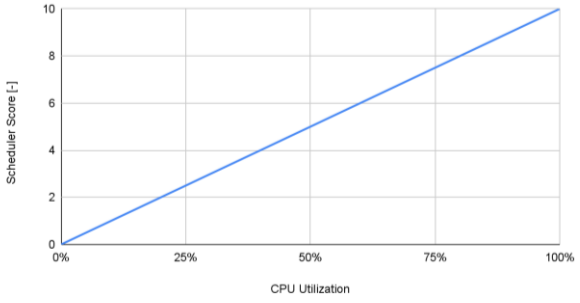



*Fig. 3. Setup 1*

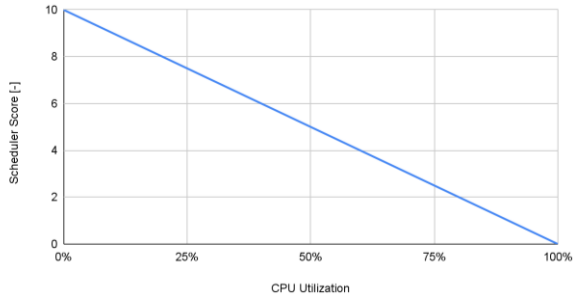



*Fig. 4. Setup 2*

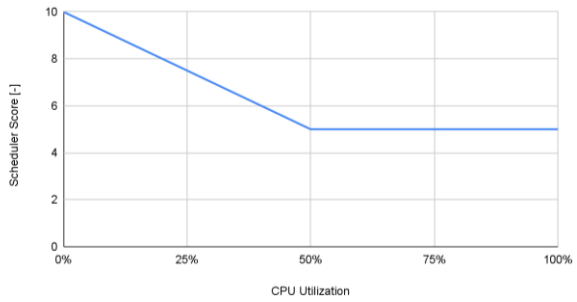



*Fig. 5. Setup 3*

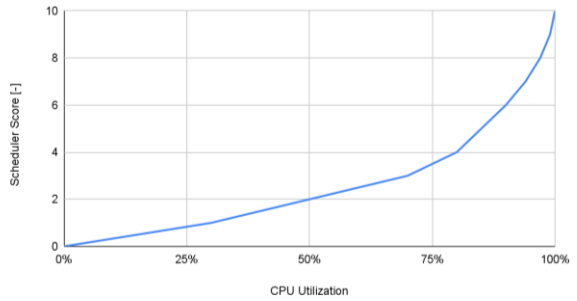



*Fig. 6. Setup 4*

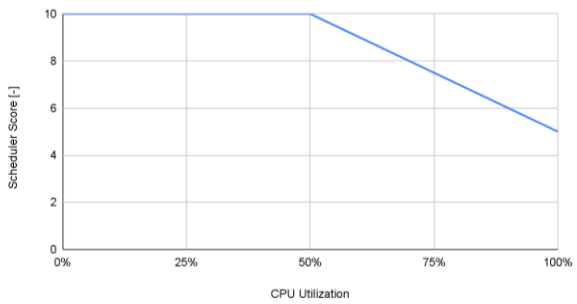



*Fig. 7. Setup 5*

The evaluated configurations are based on Kubernetes' bin-packing scheduling mechanism, implemented through the *RequestedToCapacityRatio* policy [7]. Within this framework, the Kubernetes scheduler assesses each node by comparing its current CPU utilization (i.e., requested CPU resources) against its total allocable capacity. The scheduling decision is guided by a scoring function, which maps resource utilization metrics to numerical preference scores.

In this context, the X-axis represents the CPU utilization percentage of a given node, while the Y-axis denotes the score assigned by the scheduler. This score serves as a quantitative indicator of a node's suitability for hosting additional workloads during the pod placement decision process.

Under the *RequestedToCapacityRatio* policy, the score is computed according to a predefined curve (shape) that establishes a relationship between CPU utilization (X-axis) and the resulting score (Y-axis). The *shape* parameter defines a discrete set of reference points, each representing a (utilization, score) pair. The Kubernetes scheduler performs linear interpolation between these points to derive the score for intermediate utilization values.

For instance, if the curve is defined by the points (0, 0) and (100, 10), a node operating at 50% CPU utilization would be assigned a score of approximately 5. Scores are not restricted to integer values; they may take fractional values (e.g., 4.3), yielding a continuous scoring function. The resulting interpolated curve reflects a smooth, monotonic relationship between utilization and selection priority.

A score of 10 represents the maximum selection priority, indicating that the node is most likely to be chosen to host a new pod. Conversely, a score of 4 denotes a lower selection priority – the node remains eligible but is less likely to be selected unless higher-scoring nodes are unavailable. Importantly, lower scores do not exclude a node from consideration; rather, they reduce its probabilistic weight in the scheduling decision process relative to nodes with higher scores.

Nodes with higher scores are preferentially targeted for new workload placements. In a bin-packing–oriented configuration, where the scoring function favors higher utilization, nodes already hosting workloads receive higher scores. This behavior promotes workload consolidation, improving overall resource efficiency and potentially enabling energy savings by allowing underutilized nodes to transition into low-power states.

However, this consolidation strategy introduces a performance trade-off: as nodes become more densely packed, they are more susceptible to resource contention – such as CPU throttling, memory pressure, or I/O saturation – which may negatively impact application latency, throughput stability, and overall service quality. Consequently, the optimal scoring configuration depends on the desired balance between infrastructure efficiency and runtime performance stability.

Alternative configurations that assign higher scores to less-utilized nodes mitigate contention effects and promote a more even workload distribution across the cluster. While this approach can enhance application responsiveness and predictability, it does so at the cost of reduced consolidation efficiency and higher aggregate power consumption. The choice of scoring policy therefore represents a fundamental optimization trade-off between performance isolation and resource utilization efficiency.

## 5. Results

To ensure the robustness of the experimental results, we applied several statistical aggregation methods, including interquartile range–based average (*Average IQR*), the arithmetic mean (Average), the median, and the average excluding the minimum and maximum values (Trimmed Average). The chosen statistical methods—enabled for a more accurate and reliable interpretation of the results, particularly in the presence of potential outliers.

As shown in Fig. 8, the example results for 1-node clusters obtained using the four statistical aggregation methods – Average IQR, Average, median, and Trimmed mean values – exhibit minimal variance (3–5%) across the entire spectrum of parallel execution scenarios. This consistency indicates that the performance trends observed in the experiments are not materially influenced by the choice of statistical method.

Consequently, variations in execution times can be attributed with high confidence to differences in scheduler configuration rather than to statistical noise or bias, thereby reinforcing the validity and robustness of the experimental conclusions.

The consistency across methods confirms that the dataset is not heavily skewed by extreme values and that any noise in the measurements has only a negligible effect on the aggregated metrics. Consequently, the conclusions regarding the impact of scheduler configurations on workload performance remain valid regardless of the chosen statistical method, further strengthening the credibility of the experimental findings.

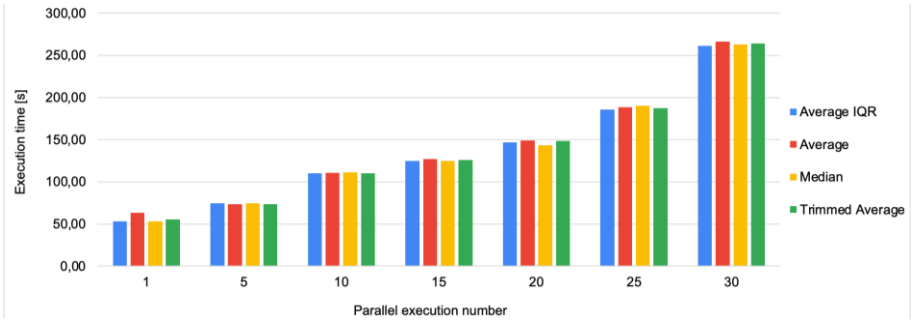



*Fig. 8. 1-node clusters execution time vs parallel execution number for no binpacking setup*

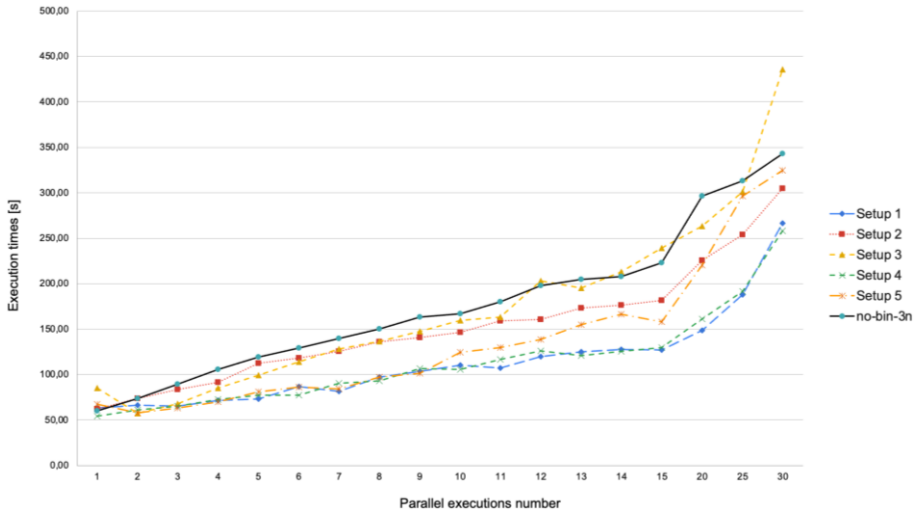



*Fig. 9. Execution Time vs parallel executions number for all setups*

It is worth noting that the minor divergence observed at higher parallelism levels (e.g., 25–30 concurrent executions) is expected, as the system experiences increased contention for CPU resources, which can lead to more variable execution times between runs. However, even in these scenarios, the statistical differences remain small enough to preserve the integrity of the results and the overall conclusions.

The empirical analysis conducted in this study demonstrates that adjustments to Kubernetes scheduler binpacking scoring functions have a measurable impact on the performance of parallel machine learning workloads, particularly under increasing concurrency.

Fig. 9. depicts execution times of parallel machine learning workloads across six different Kubernetes scheduler configurations. The horizontal axis represents the number of parallel pipeline executions (ranging from 1 to 30), while the vertical axis indicates the total execution time in seconds. Each line corresponds to a distinct scheduler setup (Setups 1–5 and baseline "no-bin-3n"), illustrating how total pipeline execution time varies with the number of concurrently executed workloads.

Initially, the performance baseline ("no-bin-3n"), representing the default Kubernetes scheduling strategy, showed a linear increase in execution time from approximately 60 seconds for single pipeline execution to nearly 340 seconds at 30 concurrent executions. This default strategy, based on the LeastAllocated scoring function, evenly distributed workloads across available nodes, resulting in predictable yet suboptimal performance due to significant inter-node communication overhead and resource fragmentation.

Comparatively, binpacking strategies using the *RequestedToCapacityRatio* plugin showed markedly distinct performance characteristics:

- Setup 1 (Linear Bin Packing): Consistently improved performance, reducing execution times from around 95 seconds at low concurrency (10 parallel executions) to approximately 270 seconds at 30 concurrent executions, effectively minimizing inter-node communication through efficient workload consolidation. This represents an improvement of roughly 20% compared to the baseline.
- Setup 2 (Least Allocated Spreading): Closely mirrored the baseline, with nearly identical performance – reaching execution times of about 335 seconds at peak concurrency – highlighting the inherent inefficiencies of dispersing closely linked ML workloads.
- Setup 3 (Flawed Spreading): Demonstrated competitive execution times below 150 seconds at low concurrency but exhibited a critical performance degradation at around 15 concurrent executions, rapidly escalating to approximately 400 seconds at higher loads due to resource saturation ("contention cliff").
- Setup 4 (Non-linear Bin Packing): Delivered optimal performance, maintaining execution times under 100 seconds at low concurrency and approximately 220 seconds at the highest concurrency tested. Its non-linear scoring function prioritized densely packed nodes intelligently, achieving both high performance and stable execution. This setup achieved up to 35% improvement over the baseline at peak concurrency.
- Setup 5 (Pathological Spreading): Demonstrated the poorest performance overall, starting at approximately 125 seconds at low concurrency and escalating dramatically beyond 325 seconds at 30 concurrent executions, driven by suboptimal pod placements and unstable scheduling behavior.

Performance volatility, quantified through the variability in execution times, was notably high for Setups 3 and 5, underlining their unsuitability for production scenarios. Conversely, Setups 1 and 4 provided predictable, stable performance vital for maintaining consistent MLOps operations and meeting service-level agreements (SLAs). Specifically, Setup 1 demonstrated a 20% performance improvement,

and Setup 4 achieved a notable 35% improvement compared to the baseline, clearly emphasizing the tangible benefits of optimized scheduling strategies.

In conclusion, the study empirically confirms that well-configured binpacking scheduling strategies, specifically leveraging the *RequestedToCapacityRatio* scoring functions, significantly enhance performance and efficiency in Kubernetes-based ML workloads at the edge.

## 6. Conclusions

The results of our experiments demonstrate that modifications to the binpacking scoring functions in the Kubernetes scheduler can noticeably influence the efficiency of running machine learning workloads on edge Kubernetes clusters – particularly under high concurrency. As the load on the system increased, the effects of scheduler configuration became more pronounced, highlighting the importance of tailoring scheduling strategies to the constraints and characteristics of edge environments.

The experimental method applied in this study – combining controlled Kubeflow pipeline executions, precise execution time measurements, and systematic variation of scheduler parameters – proved effective in isolating and quantifying the impact of scheduling strategies on performance. This approach provides a replicable framework for further research into optimizing workload placement in resource-constrained edge clusters.

While our investigation focused on CNN-based image classification using the MNIST dataset, the methodology is broadly applicable to other machine learning paradigms. As part of future work, we plan to extend experimentation to different workload types, such as clustering algorithms (e.g., K-Means, DBSCAN), Recurrent Neural Networks (RNNs) for sequence modeling, and transformer-based architectures for natural language processing. This broader scope will help determine whether the observed scheduling effects are consistent across diverse computational patterns and resource profiles.

By expanding beyond CNNs and exploring a wider variety of workloads, we aim to deepen the understanding of how scheduler characteristics interact with workload types in edge computing environments, ultimately guiding the design of more adaptive and efficient scheduling policies for real-world deployments.

## 7. Acknowledgements

This research is being undertaken as part of the 5th edition of the Implementation Doctorate program in collaboration with the Technical University of Lodz and Atos Research and Development Centre. The Polish Ministry of Education and Science financially supports the project DWD/5/0382/2021.

## References

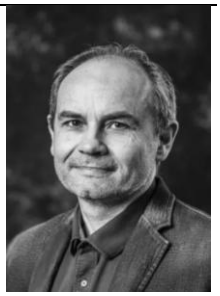
[1] Abadi M. et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2016.
[2] Almusallam N., Alabdulatif A., Alarfaj F.: Analysis of Privacy-Preserving Edge Computing and Internet of Things Models in Healthcare Domain. Comput Math Methods Med. 2021, 6834800 [https://doi.org/10.1155/2021/6834800].
[3] Chen T. et al.: MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. arXiv preprint arXiv:1512.01274, 2015.
[4] Clapa K., Grudzien K., Sierszeń A.: Assessment of Performance for Cloud-Native Machine Learning on Edge Devices. Digital Interaction and Machine Intelligence. Springer, 2024, 95–105.
[5] Guo M. et al.: K8sSim: A Simulation Tool for Kubernetes Schedulers and Its Applications. Journal of Cloud Computing, SpringerOpen, 2023.
[6] Jian Z., Wu J., Yang X.: DRS: Deep Reinforcement Learning Enhanced Scheduler for Kubernetes, Software – Practice and Experience. Wiley, 2021.
[7] Kubernetes Documentation "Resource Bin Packing". Kubernetes.io, 2025 [https://kubernetes.io/docs/concepts/scheduling-eviction/resource-bin-packing/].
[8] Kubernetes Documentation, "Resource Management for Pods and Containers". Kubernetes.io, 2025 [https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/].
[9] LeCun Y. et al.: Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE 86(11), 1998, 2278–2324.
[10] LeCun Y., Cortes C., Burges C. J. C.: The MNIST Database of Handwritten Digits [http://yann.lecun.com/exdb/mnist/].
[11] Liu P., Guitart J.: Fine-Grained Scheduling for Containerized HPC Workloads in Kubernetes Clusters. arXiv preprint arXiv:2211.11487, 2022.
[12] Lv Z. et al.: Intelligent edge computing based on machine learning for smart city. Future Generation Computer Systems 115, 2021, 90–99.
[13] Rausch T., Rashed A., Dustdar S.: Optimized container scheduling for data-intensive serverless edge computing. Future Generation Computer Systems 114, 2021, 259–271.
[14] Toka L. et al.: Machine Learning-Based Scaling Management for Kubernetes Edge Clusters. Transactions on Network and Service Management 18, 2021, 958–972.
[15] Xing E. P. et al.: Petuum: A New Platform for Distributed Machine Learning on Big Data. IEEE Transactions on Big Data 1(2), 2015, 49–67.
[16] Xiong J., Chen H.: Challenges for building a cloud native scalable and trustable multi-tenant AIoT platform. 39th International Conference on Computer-Aided Design - ICCAD'20, Article No. 26, 2020, 1–8.

**M.Sc. Konrad Clapa**
e-mail: Konrad.clap@p.dokt.pl

Konrad Cłapa is Chief Cloud Architect at Atos and a researcher at the Institute of Applied Computer Science, Lodz University of Technology. He holds a Master of Science in computer science and specializes in cloud computing, edge computing, and artificial intelligence. His research focuses on the performance and scalability of cloud-native machine learning platforms in distributed environments, and he has co-authored studies on benchmarking ML workloads with Kubernetes.
He is the author of a book on cloud computing and cloud-native architectures and holds several related patents. Konrad Cłapa is also a Google Cloud Certified Fellow and Double VMware Certified Design Expert, combining scientific research with extensive practical experience in advanced cloud architecture and implementing complex cloud architecture.

https://orcid.org/0000-0002-6939-6504

**D.Sc., Ph.D. Krzysztof Grudzień**
e-mail: artur.sierszen@p.lodz.pl



Dr. hab. inż. Krzysztof Grudzień is an associate professor at the Institute of Applied Computer Science at the Lodz University of Technology. His scientific activity focuses on the development of the field of Technical Information Technology and Telecommunications, particularly in data acquisition, processing, and analysis for non-invasive measurement and diagnostics of industrial processes. In addition, his research interests include the design of analytical algorithms and the development of functional prototypes for human–computer interaction studies. The outcomes of his work include both data processing algorithms for industrial systems and hardware designs for embedded systems incorporating sensors and actuators in wearable devices. He is the Chairman of the Human-Computer Interaction degree program and supervises the UbiCOMP Student Research Group. Dr. Grudzień has authored over 120 scientific publications and is a member of the Polish Information Processing Society.

https://orcid.org/0000-0003-4472-8100

**Ph.D. Artur Sierszen**
e-mail: artur.sierszen@p.lodz.pl

Artur Sierszeń is an assistant professor at the Institute of Applied Computer Science at the Faculty of Electrical, Electronic, Computer and Control Engineering, Lodz University of Technology. He earned a Master of Science in computer science in 2000 and a Ph.D. in computer science in 2008, specialising in artificial intelligence. Since 2000, he has worked as a researcher and lecturer at Lodz University of Technology, gaining experience in international companies such as Cisco, Infosys, Atos, and Eviden. His research interests include artificial intelligence, computer networks, and the security of both. He has authored over 50 scientific publications in these fields and holds numerous certifications in computer networks, IT security, and cloud technologies.

https://orcid.org/0000-0001-8466-4856