Impact of Resource Heterogeneity on MLOps Stages: A Computational Efficiency Study

Julio Corona[1][a], Pedro Rodrigues[1][b], Mário Antunes[1,2][c] and Rui L. Aguiar[1,2][d]
[1]*Instituto de Telecomunicações, Universidade de Aveiro, Aveiro, Portugal*
[2]*DETI, Universidade de Aveiro, Aveiro, Portugal*

Keywords: Machine Learning, Heterogeneous Computing, MLOps, DevOps.

Abstract: The rapid evolution of hardware and the growing demand for Machine Learning (ML) workloads have driven the adoption of diverse accelerators, resulting in increasingly heterogeneous computing infrastructures. Efficient execution in such environments requires optimized scheduling and resource allocation strategies to mitigate inefficiencies such as resource underutilization, increased costs, and prolonged execution times. This study examines the computational demands of different stages in the Machine Learning Operations (MLOps) pipeline, focusing on the impact of varying hardware configurations characterized by differing numbers of Central Processing Unit (CPU) cores and Random Access Memory (RAM) capacities on the execution time of these stages. Our results show that the stage involving resource-intensive model tuning significantly influences overall pipeline execution time. In contrast, other stages can benefit from less resource-intensive hardware. The analysis highlights the importance of smart scheduling and placement, prioritizing resource allocation for model training and tuning stages, in order to minimize bottlenecks and enhance overall pipeline efficiency.

1 INTRODUCTION

Machine Learning Operations (MLOps) has emerged as a critical paradigm for a streamlined development, deployment, and management of Machine Learning (ML) workflows in real-world applications (Kreuzberger et al., 2023). By integrating DevOps principles with the unique requirements of ML systems, MLOps aims to address the complexities of end-to-end ML lifecycle management (Smith, 2024). This includes tasks such as data collection, preprocessing, model training, evaluation, and deployment. While MLOps provides a structured framework to improve collaboration, scalability, and reliability, its practical implementation requires addressing several technical and computational challenges.

One of the fundamental aspects of MLOps is the orchestration of ML pipelines, which involve sequential stages designed to transform raw data into actionable insights. Each pipeline stage has different computational requirements influenced by factors such as algorithm complexity, data size, and resource availability. In practice, these pipelines are executed in heterogeneous computing environments consisting of various hardware types, including Central Processing Units (CPUs), Graphics Processing Units (GPUs), Field-Programmable Gate Arrays (FPGAs), and Tensor Processing Units (TPUs) (Faubel et al., 2023). These environments offer the potential for performance optimization by leveraging the unique strengths of different hardware components. However, achieving efficient execution in such settings requires intelligent scheduling and resource allocation strategies to avoid inefficiencies such as underutilized resources, increased costs, and prolonged execution times.

This study investigates the execution times of various ML pipeline stages within heterogeneous computing environments. It focuses on how variations in Virtual Machines (VMs) configurations, characterized by different numbers of Virtual Central Processing Units (vCPUs) cores and Random Access Memory (RAM) capacities, affect the performance of individual stages and the overall efficiency of pipeline execution in ML workflows.

The main contributions of this study are as follows: i) provides a detailed analysis of the computa-

[a] https://orcid.org/0009-0001-7898-7883
[b] https://orcid.org/0009-0009-5916-4062
[c] https://orcid.org/0000-0002-6504-9441
[d] https://orcid.org/0000-0003-0107-6253

Corona, J., Rodrigues, P., Antunes, M. and Aguiar, R. L.
Impact of Resource Heterogeneity on MLOps Stages: A Computational Efficiency Study.
DOI: 10.5220/0013520600003964
In *Proceedings of the 20th International Conference on Software Technologies (ICSOFT 2025)*, pages 246-253
ISBN: 978-989-758-757-3; ISSN: 2184-2833

tional requirements of ML pipeline stages, shedding light on how these requirements vary across tasks such as data preprocessing, feature engineering, and model training, and ii) offers insights into the impact of resource configurations, including vCPU cores and RAM, on the execution efficiency of the MLOps pipeline, emphasizing the importance of aligning hardware resources with the specific demands of different pipeline components.

The remainder of this paper is organized as follows. Section 2 provides background on MLOps and heterogeneous computing. Section 3 reviews key studies that address the challenges of deploying ML pipelines in heterogeneous environments. Section 4 outlines the research methodology, including datasets, ML pipeline stages, and computational infrastructure. Section 5 analyzes the performance of MLs pipeline stages in various VM configurations, highlighting how resource allocation affects execution times. Finally, Section 6 concludes the paper with a summary of the insights and outlines the future research directions.

2 BACKGROUND

To understand the context of this work and highlight the relevance of understanding the computational demands of different stages of the MLOps pipeline, in this section, we provide an overview of MLOps (subsection 2.1) and the challenges of running ML workloads on heterogeneous computing (subsection 2.2).

2.1 MLOps

As already mentioned, MLOps stands for Machine Learning Operations, consisting of a set of practices that automate and simplify the ML workflow and deployment. In other words, it can be seen as a culture and practice that unifies ML application development (Dev) with ML system deployment and operations (Ops). The concept of MLOps is an extension of DevOps applied to data and ML applications. Essentially, it applies the DevOps concepts relevant to the ML field and creates new practices that effectively automate the entire workflow of ML systems. By adopting MLOps practices, several benefits can be achieved, such as improved efficiency, increased scalability, reliability, enhanced collaboration, and reduced costs.

A key concept in MLOps is the ML pipeline, which consists of sequential steps covering a model's lifecycle, including data collection, data pre-processing, feature engineering, model training, model evaluation, and model deployment. Each step can be complex and time-consuming, requiring specialized tools and expertise. The use of ML pipelines helps automate and streamline these processes, enhancing reproducibility and scalability.

While somewhat abstract, Figure 1 illustrates a typical organization and flow of an MLOps workflow. As can be seen, the core component of the workflow is the ML pipeline, highlighting the importance of optimizing its execution.

MLOps has the potential to enhance ML workflows but faces several challenges, including resistance to change, lack of expertise, and organizational silos (Faubel et al., 2023). A key issue is the efficient orchestration and scheduling of ML pipelines in heterogeneous computing environments. Improper scheduling can lead to resource underutilization, wasted computational time, and higher costs. In order to develop strategies that can address these challenges, it is first essential to understand the computational requirements of different stages of the ML pipeline, which is the main focus of this work.

2.2 Heterogeneous Computing

The rapid evolution of hardware and the rise of ML workloads, which have motivated the adoption of different types of accelerators (e.g. GPUs and TPUs), have made the existing computing infrastructure heterogeneous. In this context, heterogeneous computing refers to using multiple types of hardware in a single system, such as CPUs, GPUs, FPGAs, and TPUs. Each type of hardware has its strengths and weaknesses, and by combining them in a single system, it is possible to take advantage of each other's unique capabilities to improve performance and efficiency. For example, GPUs are good at parallel processing tasks, while CPUs are better at sequential tasks. Using both types of hardware together makes it possible to achieve better performance than using either type of hardware alone. For illustration Figure 2 shows an example of a heterogeneous computing environment with CPUs, GPUs, and FPGAs.

However, as mentioned above, the benefits of heterogeneous computing may not be fully realized without proper orchestration and task placement strategies. To take full advantage of the resources available in these environments, it is crucial to develop intelligent scheduling techniques that determine the best placement of tasks that make up a workload. This is particularly important in the context of ML workloads, as different stages of a pipeline use various algorithms and techniques and, therefore, take advantage of different types of hardware. By understanding

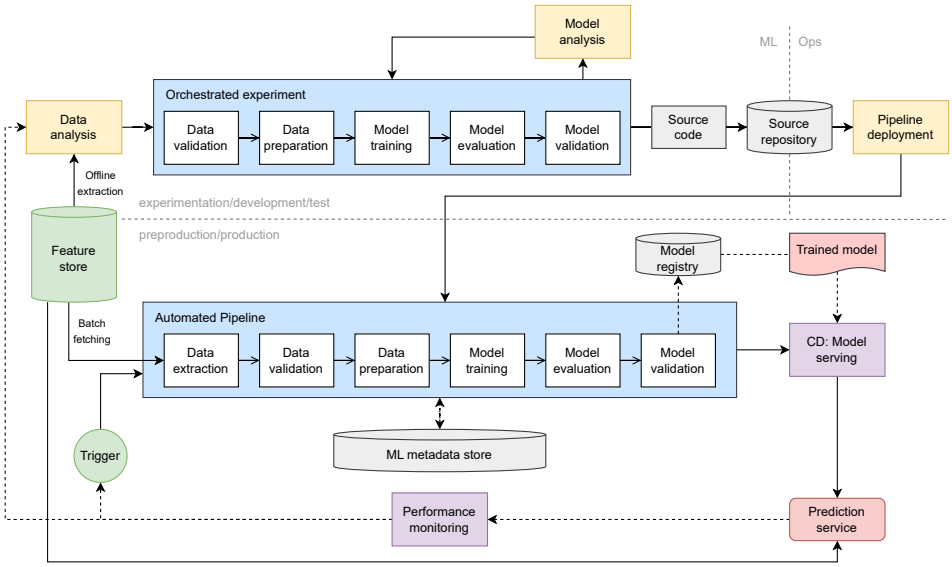

Figure 1: General overview of a common MLOps workflow. Adapted from (Google, 2024).

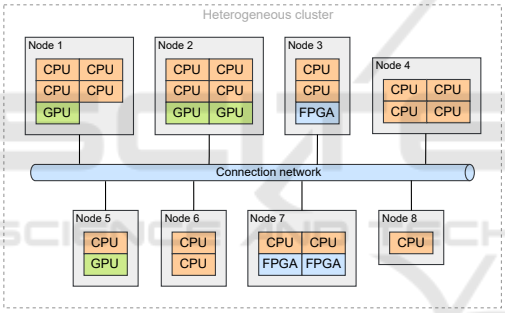

Figure 2: Simple representation of a heterogeneous cluster. Adapted from (Zhang and Wu, 2012).

the computational requirements of different pipeline stages and how they can be optimized for heterogeneous computing environments, it is possible to develop strategies to improve the efficiency and effectiveness of ML workflows.

3 RELATED WORK

Despite the novelty of the problem at hand, some authors have taken the first steps to propose solutions that mitigate the challenge of deploying ML pipelines in heterogeneous environments. One of these pioneering works is the study by N. Le et al. (Le et al., 2020), which investigates how to schedule ML jobs on interchangeable resources with different computation rates to optimize performance while ensuring fairness among users in a shared cluster. In this study, the authors propose AlloX. This framework transforms the scheduling problem into a min-cost bipartite matching problem and determines optimal CPU and GPU configurations for ML jobs, outperforming existing methods in heterogeneous setups.

Another study that attempts to approach this problem is the work of Chen et al. (Chen et al., 2023), which proposes an improved task scheduling system for Kubernetes-based MLOps frameworks using a Genetic Algorithm (GA). By modeling scheduling as a multi-objective knapsack placement problem, which is NP-hard, the authors demonstrate that their approach significantly reduces execution time compared to the default Kubernetes scheduler. Although not explicitly designed for heterogeneous environments, the fitness function considered in the GA allows the solution to be extended to this type of environment, demonstrating the potential of the proposed approach.

Motivated by hardware heterogeneity and heterogeneous performance of ML workloads, Narayanan et al. proposed Gavel (Narayanan et al., 2020), a cluster scheduler for DNN training in on-premises and cloud environments. Using a round-based scheduling mechanism, Gavel adapts existing scheduling policies to account for heterogeneity and colocation. While relevant, the work focuses only on training jobs and considers heterogeneity solely in terms of GPUs, neglecting other ML pipeline stages.

In addition, focusing only on heterogeneous GPU clusters, in (Jayaram Subramanya et al., 2023), the authors present Sia. This scheduler efficiently allocates resources to elastic resource-adaptive jobs from

heterogeneous Deep Learning (DL) clusters. The solution profiles job throughput across batch sizes on different GPU types and uses a policy optimizer for informed scheduling decisions. The authors evaluated Sia on a 64-GPU cluster with three GPU types using traces from production DL clusters, demonstrating significant reductions in average job time and makespan compared to state-of-the-art schedulers.

To achieve fair and efficient scheduling of DL workloads in heterogeneous GPU environments, Xiao Zhang proposed Mixtran (Zhang, 2024). MixTran abstracts GPU resources as virtual tickets and converts uniform requests into a global optimization model that considers resource demands, heterogeneous node constraints, and user fairness. It then applies second-price trading rules to reallocate resources, enhancing system utilization while ensuring fairness dynamically. Evaluated on a 16-GPU cluster running diverse DL jobs, MixTran reduced execution time by 30-50% compared to the default Kubernetes scheduler.

In summary, although recent work has proposed possible solutions to the problem of scheduling ML pipelines in heterogeneous environments, they have several limitations. First, most of them focus only on the training phase of ML pipelines, but a complete ML pipeline consists of several other phases, ranging from data collection to model deployment. Second, most solutions only consider heterogeneous GPU clusters, whereas, in practice, heterogeneous environments may include other types of hardware.

Although this study does not introduce novel methodologies for task allocation in diverse environments, it thoroughly explores the effects of hardware variations, specifically concerning CPU and RAM, on the execution time of ML pipelines. Examining computational demands and optimal resource configurations aims to identify key factors that can inform future research. In this context, we plan to investigate strategies for dynamic task allocation that account for hardware heterogeneity, with the goal of refining MLOps workflows and enhancing performance across diverse computing infrastructures.

4 RESEARCH METHODOLOGY

This section presents a detailed overview of the experimental framework employed in this study. It begins with an introduction to the datasets utilized, followed by a description of the computational processes involved in each stage of the ML pipeline. Experiments are conducted on diverse computational resources with varying vCPU cores and RAM configurations, allowing an analysis of how resource constraints impact the ML stages' runtime. For model tuning and evaluation, the datasets are partitioned into training and testing sets, reserving 20% of the data for testing.

4.1 Datasets

The datasets analyzed in this work span diverse domains, each possessing unique characteristics and challenges. They are sourced from publicly available repositories, with specific attributes selected for classification tasks. They include:

- **KPI-KQI** (Preciado-Velasco et al., 2021): This synthetic dataset contains 165 samples with 14 features, including Key Performance Indicators (KPIs) and Key Quality Indicators (KQIs) attributes for nine distinct services. The labels represent 5G service types.
- **User Network Activities Classification (UNAC)** (Sandeepa et al., 2023): This dataset consists of 389 samples with 23 features describing network traffic parameters such as protocols, packet sizes, and timestamps. The labels classify network activities.
- **IoT-APD** (MKhubaiib, 2024): This dataset comprises 10,485 samples with 17 features extracted from IoT simulations, including source/destination ratios and durations. The labels indicate the type of attack.
- **Network Slicing Recognition (NSR)** (Dutta,): With 31,583 samples and 17 features, this dataset includes attributes such as packet delay and equipment categories. It is designed for network slice selection, including failure scenarios.
- **DeepSlice** (Thantharate et al., 2019): This dataset includes 63,167 samples with 10 features focusing on 5G network slice parameters. The output categorizes different types of network services.
- **NSL-KDD** (Tavallaee et al., 2009): Comprising 42 attributes for 41 data characteristics and one attack label, this dataset has improved balance and diversity over the KDD-CUP dataset.
- **IoT-DNL** (Speedwall10,): This dataset contains 477,426 samples and 13 features derived from wireless IoT environments, including frame details and attack labels. Supports anomaly-based IDS evaluation.

4.2 Machine Learning Pipeline

ML pipelines are structured workflows that streamline the development, training, and evaluation of ML mod-

els. Typical ML pipelines consist of several stages, each designed to ensure effective and efficient data and model handling. These stages are essential for transforming raw data into actionable insights or predictions. Execution times at each stage were measured using Python's *exectimeit* library, which provides a wrapper specifically designed to accurately measure short execution times, as detailed in (Moreno and Fischmeister, 2017).

The methodology used in this study incorporates the following stages for developing and evaluating ML models:

Stage 1 (Data Cleaning): In this initial stage the datasets are cleaned to ensure data quality by removing rows with missing values, duplicates, and irrelevant columns.

Stage 2 (Data Preprocessing): The data preprocessing pipeline consists of multiple steps to enhance data quality and improve model performance. First, categorical variables are encoded using label encoding to convert them into numerical representations. Next, low-variance features are removed to eliminate irrelevant information. Outlier detection is performed using Isolation Forest to filter anomalous data points. The remaining features are then normalized to standardize input scales, ensuring consistency across variables. Finally, Principal Component Analysis is applied to reduce dimensionality while retaining 90% of the dataset's variance, improving computational efficiency and mitigating the risk of overfitting.

Stage 3 (Model Tuning): The stage evaluates three representatives ML models: Logistic Regression (LR), Random Forest (RF), and Multi-Layer Perceptron (MLP). Hyperparameter tuning was performed using the genetic algorithm provided by the *sklearn-genetic-opt* library. Each combination of hyperparameters was validated using 4-fold cross-validation.

Stage 4 (Model Evaluation): To evaluate the performance of the models, the Matthews Correlation Coefficient (MCC) metric (Chicco and Jurman, 2020) was selected. MCC offers a reliable and balanced assessment of classification models, particularly in scenarios involving imbalanced datasets or when evaluating performance across multiple classes is critical.

A visual representation of the ML pipeline under consideration is shown in Figure 3.

4.3 Computational Infrastructure

This experiment used nine identical computing nodes, each hosting a single VM to constrain hardware availability. The VMs were defined by their respective vCPU cores and RAM capacities, as outlined in Table 1. This setup was intentionally conservative, as many real-world frameworks exhibit greater hardware variability, including the use of dedicated accelerators such as GPUs, TPUs, and FPGAs.

Table 1: Virtual Machines Configuration.

VM	vCPU Cores	RAM (GB)
VM 1	1	1
VM 2	1	8
VM 3	1	16
VM 4	4	1
VM 5	4	8
VM 6	4	16
VM 7	8	1
VM 8	8	8
VM 9	8	16

The heterogeneous VM configuration allowed the evaluation of the feasibility and performance of the ML stages under varying resource constraints. This approach provided valuable insight into optimizing the use of computational resources to meet diverse task requirements. The heterogeneity of the experiment is simplistic (we only vary two components) for two main reasons. First, although simplistic, it allows us greater control over the possible combinations to have a better view of the impact of such small changes. Second, at the time of publication, we do not have other computational resources to allocate.

5 ANALYSIS & DISCUSSION

The following analysis examines how the pipeline stages perform in different VMs configurations and evaluates the impact of varying VMs allocations on overall ML pipeline performance.

Given the significant difference in execution times for Stage 3 compared to the other stages, Figure 4 illustrates the execution times using a logarithmic scale.

Stage 1, focused on data cleaning, demonstrated relatively consistent execution times across all configurations, with an average of approximately 0.117 seconds and a low standard deviation of 0.178 seconds. This suggests that data cleaning tasks were largely unaffected by variations in computational resources, likely due to their inherently low computational demands. Notably, VM 2 (1 vCPU and 8 GB RAM) achieved the fastest mean execution time of 0.096 seconds, outperforming VM 1 (1 vCPU and 1 GB RAM), which recorded a mean of 0.129 seconds. This indicates that increased RAM can provide a slight perfor-

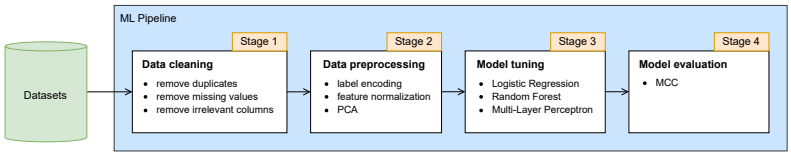

Figure 3: ML pipeline considered in this work.

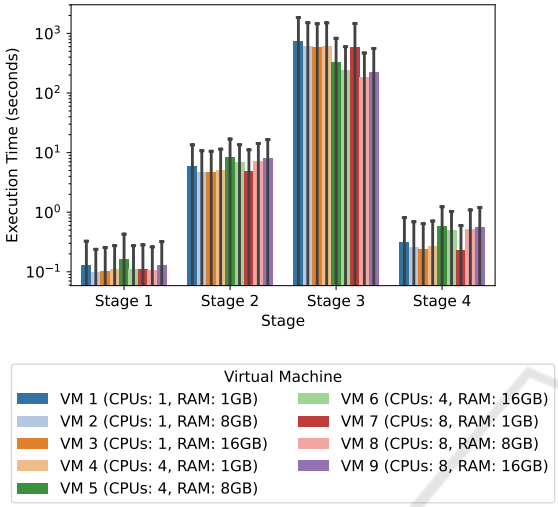

Figure 4: Execution times grouped by VM (log scale).

mance advantage even for lightweight tasks. However, the trend becomes less straightforward when considering configurations with higher vCPU counts. For instance, VM 4 (4 vCPUs and 1 GB RAM) and VM 6 (4 vCPUs and 16 GB RAM) showed comparable performance, with mean execution times of 0.111 and 0.110 seconds, respectively, suggesting that additional RAM beyond 1 GB had minimal impact. Interestingly, VM 5 (4 vCPUs and 8 GB RAM) exhibited the slowest mean execution time of 0.162 seconds, potentially indicating inefficiencies in resource allocation or overhead. Similarly, VMs with 8 vCPUs, such as VM 7 (8 vCPUs and 1 GB RAM) and VM 8 (8 vCPUs and 8 GB RAM), showed no significant performance improvement over configurations with fewer vCPUs, reinforcing the idea that excessive computational resources can lead to diminishing returns for lightweight tasks.

Stage 2, which includes data preprocessing, exhibited greater execution time variability across configurations, with mean times ranging from 4.618 seconds in VM 3 (1 vCPU, 16 GB RAM) to 8.128 seconds in VM 5 (4 vCPUs, 8 GB RAM). Notably, among single-vCPU configurations, increasing RAM consistently improved performance, as seen in VM 3 achieving the lowest mean execution time. However, for multi-vCPU configurations, performance trends were less predictable, with VM 5 (4 vCPUs, 8 GB RAM) and VM 9 (8 vCPUs, 16 GB RAM) experiencing the highest mean execution times. Interestingly, VM 7 (8 vCPUs, 1 GB RAM) outperformed several higher-resource configurations, suggesting inefficiencies in resource management at higher core and memory levels. The results highlight that for this stage, while additional RAM can enhance performance in single-core environments, increased CPU and memory allocations do not always yield proportional benefits for this stage.

Stage 3, focused on model tuning, showed a significant reduction in execution time as computational resources increased, with VM 8 (8 vCPUs, 8 GB RAM) achieving the lowest mean time of 184.36 seconds. Single-vCPU configurations exhibited the highest execution times, with VM 1 (1 vCPU, 1 GB RAM) reaching 734.25 seconds, highlighting the limitations of minimal resources for this stage. Among multi-vCPU configurations, VM 6 (4 vCPUs, 16 GB RAM) and VM 8 demonstrated the best performance, suggesting that both increased CPU cores and sufficient RAM contribute to efficiency. However, VM 7 (8 vCPUs, 1 GB RAM) showed worse performance than VM 5 (4 vCPUs, 8 GB RAM), indicating that memory constraints can bottleneck execution even with higher CPU counts. The results suggest a balanced combination of CPU and RAM is necessary to achieve optimal performance at this stage.

Stage 4, which involved model evaluation and selection, was less resource-intensive than model tuning but showed varying performance across different configurations, with VM 7 (8 vCPUs, 1 GB RAM) achieving the fastest mean execution time of 0.231 seconds, followed closely by VM 3 (1 vCPU, 16 GB RAM) with 0.238 seconds. Single-vCPU configurations like VM 1 (1 vCPU, 1 GB RAM) showed slightly higher mean times of 0.306 seconds, while those with additional RAM, such as VM 2 (1 vCPU, 8 GB RAM), demonstrated improved performance with 0.255 seconds. The performance of multi-vCPU configurations was less consistent; for example, VM 4 (4 vCPUs, 1 GB RAM) performed at 0.262 seconds, while VM 6 (4 vCPUs, 16 GB RAM) took 0.490 seconds, showing that adding CPUs and RAM may not always lead to better results.

When analyzing how VMs allocations influence the total pipeline execution, all possible combinations

were evaluated. The distribution of these times is shown in Figure 5. Each combination is denoted as $[S_1, S_2, S_3, S_4]$, where S_i represents the VM assigned to Stage i. Although our pipeline follows a typical structure, it does not include all possible operations for each stage. As a result, the distribution could exhibit different properties, but the minimum and maximum execution times would remain the same.

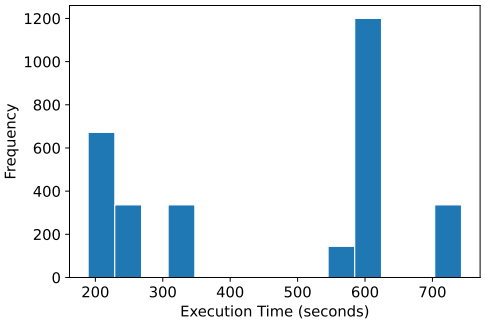

Figure 5: Pipeline execution time histogram.

A notable observation is the pronounced clustering of execution times at both ends of the histogram. The interval from 184.363 to 218.755 seconds shows a frequency count of 672, while the interval between 578.427 and 602.224 seconds peaks at 1008 occurrences. This indicates that numerous VMs combinations result in highly efficient execution times or are grouped towards the upper limit of the execution times. The mean and standard deviation of the total execution times, 458.960 seconds and 196.416 seconds, respectively, reveal that most VM allocations are suboptimal compared to the best total time of 184.363 seconds, while the worst total time of 734.250 seconds is approximately 1.4 standard deviations above the mean. These metrics underscore the expected performance of simpler allocation strategies, such as Round Robin or First In, First Out (FIFO), which are likely to produce results near the average due to their lack of workload-specific optimization.

The first group represents the best-performing configurations, yielding the shortest total execution times and highlighting their efficiency in pipeline execution. In particular, the combinations [VM 2, VM 3, VM 8, VM 7], [VM 6, VM 3, VM 8, VM 7], and [VM 4, VM 3, VM 8, VM 7] produced the shortest total times, all approximately 189.317 seconds. These configurations used a mixture of VMs with seemingly optimized resource allocations for the pipeline stages, particularly stage 3, which dominates the total execution time. The consistently low execution times in these configurations suggest that VM 8 is highly suited for the resource-intensive Stage 3, while VMs 2, 3, 4, or 7 effectively handle the less demanding earlier stages.

In contrast, other groups of combinations exhibited total times ranging from 228.860 seconds to 347.516 seconds. These configurations often involved using VMs 5 or 6 for Stage 3, which proved to be less efficient than VM 8 for handling this resource-heavy stage. Furthermore, slight variations in total times within this range indicate the interaction of VMs assignments for earlier stages, where minor inefficiencies can compound when paired with a suboptimal Stage 3 configuration.

The worst-performing combinations had total times ranging from 545.275 to 624.379 seconds and from 703.483 to 743.035 seconds. The first group, between 545.275 and 624.379 seconds, included combinations such as [VM 8, VM 2, VM 3, VM 7], [VM 4, VM 2, VM 3, VM 7], [VM6, VM5, VM4, VM9], and [VM3, VM5, VM7, VM8]. These configurations assigned Stage 3 to VMs 2, 3, 4, or 7, which lacked sufficient resources to process efficiently, causing significant bottlenecks. The second group, between 703.483 and 743.035 seconds, contained combinations like [VM 7, VM 5, VM 1, VM 9], which not only assigned Stage 3 to an inefficient VM (VM 1) but also compounded inefficiencies by suboptimal assignments in earlier stages. This highlights the importance of ensuring that the most resource-intensive stages are handled by VMs with sufficient capacity to avoid cascading inefficiencies.

6 CONCLUSION AND FUTURE DIRECTIONS

This study highlights the critical role of appropriate VM configurations in optimizing ML pipeline performance, particularly for stages with longer processing times. Our analysis demonstrates that the performance of Stage 3, which involves model tuning, has the most substantial impact on the total pipeline execution time. Configurations utilizing VM 8 consistently achieved better performance for Stage 3. In contrast, combinations involving VMs with limited resources (for example, VM 1 or VM 4 with 1 GB of RAM) caused significant delays.

Furthermore, selecting less resource-intensive VMs for earlier stages proved beneficial in optimizing performance. For example, configurations such as [VM 2, VM 3, VM 8, VM 7], [VM 6, VM 3, VM 8, VM 7], and [VM 4, VM 3, VM 8, VM 7] delivered the shortest total execution times by balancing resource allocation across all stages. In contrast, configurations that paired suboptimal VMs for Stage 3 with other stages caused inefficiencies, contributing

to longer execution times.

This study underscores the importance of prioritizing resource allocation, particularly for the most resource-intensive stages, to minimize bottlenecks and ensure efficient ML pipeline execution. However, since many real-world frameworks exhibit greater hardware variability, including the use of dedicated accelerators such as GPUs, TPUs, and FPGAs, future research should investigate whether these findings can be generalized to such environments. Additionally, future work should explore dynamic VM provisioning and adaptive resource management strategies to enhance performance in heterogeneous and evolving computing scenarios.

ACKNOWLEDGEMENTS

This study was funded by the PRR – Plano de Recuperação e Resiliência and by the NextGenerationEU funds at University of Aveiro, through the scope of the Agenda for Business Innovation "NEXUS: Pacto de Inovação – Transição Verde e Digital para Transportes, Logística e Mobilidade" (Project nº 53 with the application C645112083-00000059).

REFERENCES

Chen, H.-M., Chen, S.-Y., Hsueh, S.-H., and Wang, S.-K. (2023). Designing an improved ml task scheduling mechanism on kubernetes. In *2023 Sixth International Symposium on Computer, Consumer and Control (IS3C)*, pages 60–63.

Chicco, D. and Jurman, G. (2020). The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):1–13.

Dutta, G. Network slicing recognition. https://www.kaggle.com/datasets/gauravduttakiit/network-slicing-recognition.

Faubel, L., Schmid, K., and Eichelberger, H. (2023). Mlops challenges in industry 4.0. *SN Computer Science*, 4(6):828.

Google (2024). Mlops: Continuous delivery and automation pipelines in machine learning. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

Jayaram Subramanya, S., Arfeen, D., Lin, S., Qiao, A., Jia, Z., and Ganger, G. R. (2023). Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 642–657, New York, NY, USA. Association for Computing Machinery.

Kreuzberger, D., Kühl, N., and Hirschl, S. (2023). Machine learning operations (mlops): Overview, definition, and architecture. *IEEE access*, 11:31866–31879.

Le, T. N., Sun, X., Chowdhury, M., and Liu, Z. (2020). Allox: compute allocation in hybrid clusters. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA. Association for Computing Machinery.

MKhubaiib. IoT attack prediction dataset. https://www.kaggle.com/datasets/mkhubaiib/iot-attack-prediction-dataset.

Moreno, C. and Fischmeister, S. (2017). Accurate measurement of small execution times—getting around measurement errors. *IEEE Embedded Systems Letters*, 9(1):17–20.

Narayanan, D., Santhanam, K., Kazhamiaka, F., Phanishayee, A., and Zaharia, M.(2020). Heterogeneity-Aware cluster scheduling policies for deep learning workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 481–498. USENIX Association.

Preciado-Velasco, J. E., Gonzalez-Franco, J. D., Anias-Calderon, C. E., Nieto-Hipolito, J. I., and Rivera-Rodriguez, R. (2021). 5g/b5g service classification using supervised learning. *Applied Sciences*, 11(11):4942.

Sandeepa, C., Senevirathna, T., Siniarski, B., Nguyen, M.-D., La, V.-H., Wang, S., and Liyanage, M. (2023). From opacity to clarity: Leveraging xai for robust network traffic classification. In *International Conference on Asia Pacific Advanced Network*, pages 125–138. Springer.

Smith, J. (2024). Devops and mlops convergence: Improving collaboration between data science and engineering teams. *Australian Journal of Machine Learning Research & Applications*, 4(2):82–86.

Speedwall10. Iot device network logs. https://www.kaggle.com/datasets/speedwall10/iot-device-network-logs.

Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. Ieee.

Thantharate, A., Paropkari, R., Walunj, V., and Beard, C. (2019). Deepslice: A deep learning approach towards an efficient and reliable network slicing in 5g networks. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0762–0767. IEEE.

Zhang, K. and Wu, B. (2012). Task scheduling for gpu heterogeneous cluster. In *2012 IEEE International Conference on Cluster Computing Workshops*, pages 161–169.

Zhang, X. (2024). Mixtran: an efficient and fair scheduler for mixed deep learning workloads in heterogeneous gpu environments. *Cluster Computing*, 27(3):2775–2784.