Optimised Task Placement for MLOps

Pedro Rodrigues*, Leonardo Almeida*, Julio Corona*, Mário Antunes*† and Rui L. Aguiar*†
*Instituto de Telecomunicações, Universidade de Aveiro, Aveiro, Portugal
E-mails: {pedrofrodrigues,leonardoalmeida,jcamejo,mario.antunes,ruilaa}@av.it.pt
†DETI, Universidade de Aveiro, Aveiro, Portugal
E-mails: {mario.antunes,ruilaa}@ua.pt

***Abstract*—To accelerate Machine Learning Operations (MLOps) workflows, where Machine Learning (ML) pipelines are the most time-consuming element, heterogeneous environments can be used. However, to take full advantage of this heterogeneity, it is essential to determine the appropriate machine for each task in the pipeline. In this paper, we present a task placement system for ML pipelines that automatically assigns tasks to the most appropriate machine, with the goal of reducing the overall execution time. The placement process is divided into two phases: pipeline scheduling and task placement. The scheduling phase uses the Shortest Job First (SJF) algorithm to determine the pipeline execution order, while the task placement phase uses a heuristic-based approach. Results show that the proposed solution significantly outperforms baseline strategies, achieving reductions in total execution time and average waiting time.**

***Index Terms*—Machine Learning, MLOps, Heterogeneous Computing, Task placement**

I. Introduction

To meet the growing demand for Machine Learning (ML) based solutions, Machine Learning Operations (MLOps) has gained popularity as a methodology for automating and streamlining the entire ML lifecycle, from model development to production deployment. A key element of an MLOps workflow is the ML pipeline, which consists of a sequence of tasks performed in a specific order to build, train and evaluate a model. As this phase is often the most time-consuming and computationally intensive part of the workflow, speeding up its execution becomes critical.

One effective approach to achieve this is to exploit the heterogeneity of today's computing environments, which provide a variety of computing nodes equipped with different hardware resources, such as Central Processing Units (CPUs), and Graphics Processing Units (GPUs). However, to take full advantage of this heterogeneity, it is crucial to automatically and intelligently assign each task of the ML pipeline to the appropriate machine [1]. Currently, existing MLOps platforms do not provide this capability, relying on manual configuration or traditional algorithms to assign tasks to machines.

In this paper, in the context of MLOps, we propose a task placement system for ML pipelines that automatically assigns tasks to the most appropriate machine. By doing so, we aim to reduce the overall execution time of the pipelines and therefore accelerate the development of ML models.

The remaining of this document is organised as follows: Section 2 presents the current efforts to address the problem at hand, Section 3 describes the proposed solution, and Section 4 presents the experimental methodology. Finally, Section 5 presents the obtained results, and Section 6 concludes the paper and discusses future work.

II. Related Work

Despite the novelty of the problem, some authors have taken initial steps to mitigate the challenges of deploying ML pipelines in heterogeneous environments. These solutions vary in approach, but generally rely on heuristics, task profiling, and machine learning techniques.

For example, [2] and [3] estimate task execution times by profiling tasks on different devices using small data samples. Specifically, [2] uses a min-cost bipartite matching algorithm to assign tasks to devices, while [3] uses a heuristic-based policy. To reduce the profiling overhead, some authors have proposed ML-based solutions. For instance, [4] uses two supervised models: one to predict device suitability and another to estimate execution time, which feeds a heuristic task assignment algorithm. In contrast, [5] introduces a reinforcement learning agent that captures task-resource interdependencies and optimises task assignment based on a reward function that considers task execution time.

Despite these progresses, significant challenges remain. Most existing methods focus only on the training phase, whereas a complete ML pipeline includes data collection through model evaluation. Task profiling techniques suffer from scalability issues, and supervised ML-based methods require high-quality training data, which is often scarce, reducing their robustness. In addition, none of these solutions integrate seamlessly with existing MLOps platforms, limiting their practical applicability in production environments.

III. Proposed solution

To overcome the limitations of existing solutions and bridge the gap in current MLOps platforms, we propose a task placement system for ML pipelines. The system automatically assigns each pipeline task to the most suitable machine to minimise the overall execution time (makespan). These tasks cover the entire ML lifecycle, from data validation to model evaluation. The following subsections describe the architecture of the system, and the placement strategy adopted.

A. System architecture

To ensure scalability and robustness, the system follows a modular architecture that seamlessly integrates with existing

MLOps platforms. As shown in Figure 1, the architecture consists of five main components: (i) pipeline manager, (ii) cluster node manager, (iii) model manager, (iv) data manager, and (v) placement decision unit. The pipeline manager accepts ML pipelines from users, decomposes them into processes and triggers their execution. The cluster node manager monitors the cluster nodes and their hardware resources. The model manager provides detailed information about a specific ML algorithm, including the estimated total number of operations to be performed, both for training and evaluation phases. The data manager analyses the pipeline's input data, including its size, type and format. Finally, the placement decision unit receives information from the other components and decides in which order the pipelines should be executed and where to place each task.

A common practice in MLOps is to containerise each task in the pipeline and orchestrate them using container management tools. As this aspect is beyond our scope, we use an existing MLOps platform, such as Kubeflow, to handle the orchestration. In this way, the system acts as an intermediary between the users and the MLOps platform, providing the necessary information for efficient task execution.

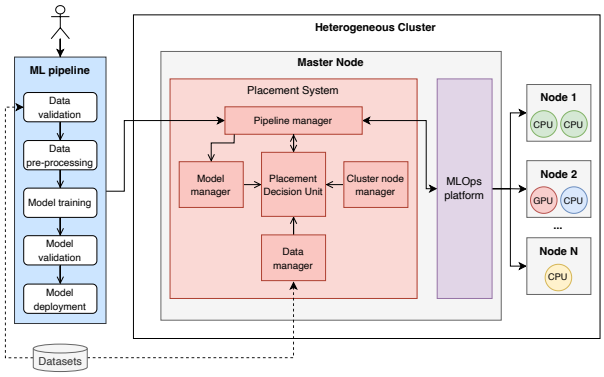


Fig. 1: Solution architecture.

B. Placement Strategy

Since ML clusters are typically shared among multiple users, it is important not only to minimise pipeline execution times, but also to reduce waiting times (the delay between submission and the start of execution). To address both objectives, the proposed placement strategy is divided into two distinct phases: pipeline scheduling, and task placement.

In the first phase, to allow the system to select an optimal execution order, multiple pipeline submissions are accumulated during a configurable time window. Once the window closes, the system determines an execution order that minimises overall waiting time. The scheduling algorithm used is Shortest Job First (SJF), which prioritises pipelines with the shortest execution time. By executing shorter pipelines first, longer pipelines do not block shorter ones, resulting in reduced average waiting times.

Accurately predicting the execution time of a pipeline is challenging due to variability in ML algorithms, hyperparameters, input data, and available hardware. To address this, the system assumes the length of a pipeline as the sum of the total number of operations required for each task in the pipeline. This approach is based on the assumption that the execution time of a task is proportional to the number of operations it performs. The calculation of the total number of operations across all tasks in the pipeline is derived from model parameters, data characteristics, and the theoretical computational complexity of the ML algorithms used. Although theoretical complexity may overestimate the actual number of operations performed, it provides a reasonable approximation for effective scheduling.

Once the execution order is established, the second phase assigns each task to the most suitable machine. The task placement uses a heuristic-based approach that considers: (i) input data characteristics, (ii) ML model family and parameters, (iii) current machine load, and (iv) hardware resource availability.

For preprocessing tasks, the system estimates memory requirements using user-provided dataset details (e.g., number of samples, features, and feature types). It filters for nodes with sufficient memory to load the dataset plus a 50% overhead for processing. Among the eligible nodes, the one with the fewest assigned tasks is chosen to promote load balancing across the cluster.

For training tasks, the system evaluates the ML model family, model size, and dataset size. The mapping is based on the following criteria:

- Linear models and simple tree-based models are mapped to low and medium performance machines due to their relatively low computational demands.
- Support Vector Machines (SVMs) and ensemble models are assigned to medium and high performance machines.
- Deep learning models are mapped to high performance machines, as they benefit significantly from multicore CPUs.

The selected node must also be lightly loaded and have sufficient memory to accommodate the training data.

For evaluation tasks, the procedure is similar to training, but the computational load is typically lower. Here, mapping is based on the model size and test dataset size:

- Linear models, SVMs, and ensemble models are assigned to low and medium performance machines.
- Deep learning models are mapped to medium and high performance machines.

Once task scheduling and placement is complete, the system triggers pipeline execution. To ensure that different pipelines do not compete for the same resources and to prevent resource contention, the system forces each node to execute a single task at a time. This means that if a node is already executing a task, it will not accept new tasks until the current task has been completed. However, a limitation of this approach is that some nodes assigned to a pipeline may remain idle while waiting for a task on another node to complete. To mitigate this problem, the placement strategy attempts to reuse the same node for tasks that are part of the same pipeline, as long as the node has

the required resources available, allowing multiple pipelines to run simultaneously without pipelines competing for the same resources.

IV. Experimental methodology

To evaluate the performance of the implemented system, we conducted an experiment in which multiple ML pipelines were submitted to the system in the same time window. In order to understand the impact of the proposed solution, several baseline strategies were also implemented. The following subsections describe the experimental setup, the ML pipelines used, and the baseline strategies used for comparison.

A. Experimental setup

The experiment were performed on a computing cluster with 10 nodes, 9 of which are worker nodes responsible for executing the tasks, while the remaining node is the master node that manages the cluster and coordinates the execution of the pipelines. In terms of heterogeneity, three different types of worker nodes were considered: low, medium and high performance nodes. Each type of node has different hardware specifications, including the amount of Random Access Memory (RAM), number of CPU cores, and CPU architecture. Each node in the cluster and its specifications are listed in Table I.

TABLE I: Worker nodes considered in the experiment.

Worker nodes	CPU architecture	CPU cores	RAM
ml-worker-low-01	amd64	2	2
ml-worker-low-02	amd64		
ml-worker-low-03	arm64		
ml-worker-med-01	amd64	4	8
ml-worker-med-02	amd64		
ml-worker-med-03	arm64		
ml-worker-high-01	amd64	8	16
ml-worker-high-02	amd64		
ml-worker-high-03	arm64		

B. ML pipelines

The ML pipelines used to evaluate the system consist of three sequential tasks: data preprocessing, model training, and model evaluation. The data preprocessing step involves loading the dataset, performing necessary transformations or feature engineering, and splitting the data into training and test sets. The model training step focuses on training the model using the training set, while the model evaluation step assesses the trained model using the test set.

For the experiment, five commonly used classification datasets were selected: the Adult Income dataset [6], UNSW-NB15 [7], KDD Cup 99 [8], MNIST [9], and CIFAR-10 [10]. These datasets were chosen to cover a wide range of characteristics, including variations in dataset size, feature types, and complexity levels.

A total of eight distinct ML pipelines were created: three pipelines using the Adult dataset, two pipelines using the KDD Cup 99 dataset, and one pipeline for each of the remaining datasets. The dataset and model for each pipeline are presented in Table II. These pipelines were designed to incorporate a variety of ML algorithms, including linear models, decision trees, and deep learning models.

TABLE II: Pipelines used for evaluation.

Pipeline	ML model	Dataset
1	Logistic regression	Adult income
2	Decision tree	
3	Random forest	
4	Random forest	kddcup99
5	SVM	
6	SVM	UNSW-NB15
7	DNN	MNIST
8	CNN	CIFAR 10

C. Baselines

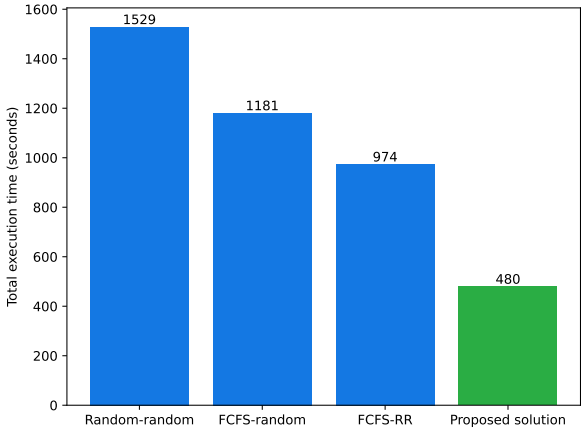
To evaluate the performance of the proposed system, three additional strategies were implemented as baselines. Similar to the proposed strategy, these strategies also include the scheduling and placement phases. In the first baseline strategy, both scheduling and placement are performed randomly, without taking into account the characteristics of the pipelines or the cluster. In the second strategy, scheduling is carried out using the First-Come First-Served (FCFS) algorithm, where pipelines are executed in the order they are submitted. However, placement in this case is random. In the third strategy, the scheduling also follows the FCFS approach, but placement follows the Round Robin (RR) algorithm, where tasks are assigned to nodes circularly.

V. Results

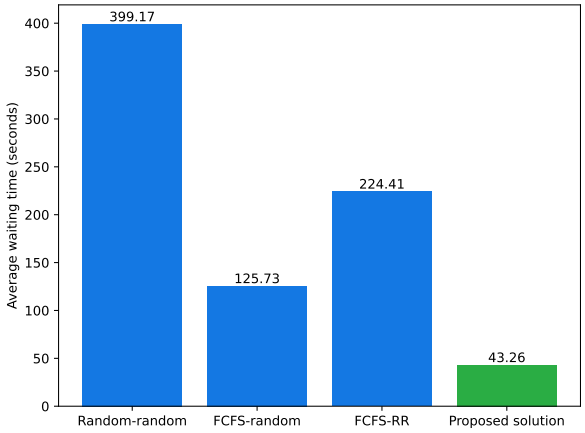
To evaluate the proposed solution, we measured two key metrics: total execution time and average waiting time. The total execution time is defined as the time taken to execute all submitted pipelines, from the moment the first pipeline is submitted until the last pipeline is completed. In contrast, the average waiting time is the average time each pipeline waits before starting execution after being submitted. For the baseline strategies involving randomness, we conducted three separate runs and calculated both the average execution time and the average waiting time. The order in which the pipelines were submitted was also randomised, and the same order was used for all strategies. The results of the experiments are shown in Figure 2a and Figure 2b.

As shown in both figures, the proposed solution consistently outperforms all baseline strategies in terms of total execution time and average waiting time.

For total execution time, the purely random strategy performed the worst, taking 3.19 times longer than the proposed solution. The FCFS-random and FCFS-RR strategies were also slower, with execution times 2.46 and 2.03 times higher, respectively. This translates to reductions of 68.61%, 59.36%,



(a) Total execution time to complete the pipelines.



(b) Average waiting time of the pipelines.

Fig. 2: Performance comparison of baseline strategies and the proposed solution. "Random-random" uses random scheduling and placement, "FCFS-random" combines FCFS scheduling with random placement, and "FCFS-RR" uses FCFS scheduling with Round-Robin placement.

and 50.72% in total execution time compared to the random, FCFS-random, and FCFS-RR strategies.

Similarly, for average waiting time, the proposed solution achieved significant improvements. Compared to the proposed approach, the purely random, FCFS-random, and FCFS-RR strategies had waiting times that were 9.23, 2.91, and 5.19 times higher, corresponding to reductions of 89.16%, 65.59%, and 80.72%, respectively.

These results demonstrate that prioritizing pipeline execution based on estimated length, assigning tasks to the most suitable machines, and try to reuse the same node for tasks within the same pipeline significantly reduces both execution and waiting times.

VI. Conclusion

This paper presents a task placement system designed to optimise the execution of ML pipelines within MLOps workflows. The strategy adopted starts by scheduling pipelines based on their estimated total number of operations, which is calculated from the theoretical computational complexity of the ML algorithms used and the characteristics of the input data. To further improve performance, the system follows a heuristic-based approach to assign each task to the most appropriate machine, taking into account the characteristics of the ML model, the input data, and the current load of the machines in the cluster.

Experimental results confirmed the effectiveness of our solution, showing significant performance gains in terms of pipeline completion time and average waiting time compared to baseline strategies. The proposed solution outperformed the random, FCFS-random, and FCFS-RR strategies by 68.61%, 59.36%, and 50.72% in total execution time, and by 89.16%, 65.59%, and 80.72% in average waiting time, respectively.

Future work includes enhancing the system with adaptive learning capabilities that improve task placement decisions over time, for example using reinforcement learning techniques. We also plan to add support for more level of heterogeneity, such as GPUs.

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

[1] D. Xin, H. Miao, A. Parameswaran, and N. Polyzotis, "Production machine learning pipelines: Empirical analysis and optimization opportunities," in *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, (New York, NY, USA), pp. 2639–2652, Association for Computing Machinery, 2021.

[2] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "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, 2020.

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

[4] U. Ahmed, M. Aleem, Y. Khalid, M. A. Islam, and M. Iqbal, "Ralb-hc: A resource-aware load balancer for heterogeneous cluster," *Concurrency and Computation: Practice and Experience*, vol. 33, p. e5606, 12 2019.

[5] M. Cheong, H. Lee, I. Yeom, and H. Woo, "Scarl: Attentive reinforcement learning-based scheduling in a multi-resource heterogeneous cluster," *IEEE Access*, vol. 7, pp. 153432–153444, 2019.

[6] B. Becker and R. Kohavi, "Adult." UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[7] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6, 2015.

[8] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan, "KDD Cup 1999 Data." UCI Machine Learning Repository, 1999. DOI: https://doi.org/10.24432/C51C7N.

[9] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[10] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images.," 2009.