Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Workload-Aware Performance Evaluation of Sequential and Parallel DAG-Based Machine Learning Orchestration on Single-Node Systems

Krisna Shaadiq Nugroho[1)], Rama Aria Megantara [2) *], Farrikh Al Zami [3)], Firman Wahyudi [4)], Ricardus Anggi Pramunendar [5)], Novita Kurnia Ningrum [6)], Chaerul Umam [7)], L.Budi Handoko [8)]
[1,2,3,4,5,6,7,8)] Information Engineering, Faculty of Computer Science, Universitas Dian Nuswantoro, Indonesia
[1)]111202214776@mhs.dinus.ac.id, [2)] aria@dsn.dinus.ac.id, [3)] alzami@dsn.dinus.ac.id, [4)] firman@dinustech.com
[5)] ricardus.anggi@dsn.dinus.ac.id, [6)] novita.kn@dsn.dinus.ac.id [7)] chaerul@dsn.dinus.ac.id,
[8)] handoko@dsn.dinus.ac.id

Submitted :Jan 6, 2026 | **Accepted** : Jn 29, 2026 | **Published** : Jan 30, 2026

Abstract: The increasing adoption of machine learning in production systems has intensified the need for structured, automated, and reproducible pipelines, commonly modeled as directed acyclic graphs. However, unlike most existing workflow scheduling studies that focus on distributed or multi-node environments, this work addresses the lack of controlled, workload-aware analysis for orchestration strategies specifically in single-node systems. A controlled experimental methodology is applied by executing an identical machine learning pipeline under two orchestration modes: sequential execution using a local orchestrator and parallel execution using a workflow orchestration engine. Two scenarios are evaluated by explicitly controlling task execution duration to represent light and heavy computational workloads. The results show that under light workloads, parallel execution increases the average makespan significantly, yielding a speedup of only 0.71, which indicates performance degradation due to dominant orchestration overhead. In contrast, under heavy workloads, parallel execution reduces the average makespan from 1013 seconds to 532 seconds, achieving a speedup of 1.90. System-level monitoring reveals higher central processing unit utilization during parallel execution, while evaluation metrics, including root mean square error and coefficient of determination, remain stable across all experimental runs. This study contributes empirical evidence of a workload threshold beyond which orchestration overhead becomes negligible and pipeline parallelism becomes beneficial. These findings demonstrate that the performance benefits of parallelism are strictly workload-dependent, highlighting the importance of selecting orchestration strategies based on computational workload characteristics.

Keywords: DAG scheduling; Machine learning pipelines; Orchestration overhead; Parallel execution; Single-node environment

INTRODUCTION

The rapid advancement of machine learning (ML) has driven an increasing demand for structured, automated, and reproducible computational pipelines, particularly in the context of model development and operationalization within modern Machine Learning Operations (MLOps) practices (Kramer & Lu, 2025; Nelavelli & Augie, 2022; Steidl et al., 2023). ML pipelines are commonly represented as Directed Acyclic Graphs (DAGs), where each node corresponds to a processing stage and each edge represents inter-stage dependencies, enabling explicit modeling of computational workflows and controlled execution (Albtoush et al., 2023; Verucchi et al., 2023). DAG-based approaches have been widely adopted across scientific workflows, large-scale data processing, and parallel computing systems due to their ability to formally describe task dependencies while exposing opportunities for parallel execution(Davami et al., 2021; Eladgham et al., 2024; Verucchi et al., 2023).

As the adoption of ML in production environments continues to increase, numerous ML pipeline orchestration platforms have been developed to manage the complexity of the end-to-end ML lifecycle, including data

* Rama Aria Megantara

e-ISSN : 2541-2019
p-ISSN : 2541-044X

preprocessing, model training, evaluation, and deployment (Kramer & Lu, 2025; Steidl et al., 2023). These platforms typically provide automation capabilities, experiment tracking, and integration with container-based infrastructure and workflow engines to enhance reproducibility and scalability (Nelavelli & Augie, 2022; Theusch & Heisterkamp, 2024; Yasmin et al., 2025). However, the use of orchestration systems introduces additional system-level overhead, such as service initialization, task scheduling, and workflow coordination, which can adversely affect execution performance, particularly in environments with limited computational resources (Corodescu et al., 2021; Elshamy et al., 2023; Yasmin et al., 2025).

In the context of DAG scheduling and parallel systems, parallel execution is commonly assumed to improve performance relative to sequential execution, especially with respect to makespan reduction and resource utilization (Davami et al., 2021; Eladgham et al., 2024; Stewart et al., 2023). This assumption generally holds in distributed or multi-node environments with abundant computational resources, where orchestration costs are relatively small compared to core computation time (Davami et al., 2021; Hu et al., 2023). In contrast, in resource-constrained scenarios such as single-node environments with limited CPU and memory capacity, orchestration overhead may dominate the total execution time, particularly when the executed pipelines are computationally lightweight (Corodescu et al., 2021; Elshamy et al., 2023).

However, a critical gap remains in the current literature. Although extensive studies have investigated DAG scheduling and workflow optimization, most empirical evaluations focus on multi-node or distributed environments. There is still limited experimental evidence on how orchestration overhead interacts with workload characteristics in single-node ML pipelines, particularly in modern MLOps frameworks. Consequently, the influence of workload characteristics on the effectiveness of pipeline parallelism has not been sufficiently examined in practical orchestration settings.

This study addresses this limitation by offering a workload-aware performance analysis. This study is novel in that it provides a controlled experimental investigation of orchestration strategies by explicitly isolating orchestration overhead from algorithmic complexity in a single-node environment. It focuses on the behavior of modern orchestration mechanisms in managing inter-task parallelism under different workload characteristics and evaluates their impact on makespan, system resource utilization, and the stability of machine learning training outcomes (Kramer & Lu, 2025; Yasmin et al., 2025).

To systematically guide this evaluation, this study addresses the following three research questions:

- RQ1: Does parallel orchestration always improve makespan in single-node ML pipelines?
- RQ2: How does workload intensity affect orchestration efficiency?
- RQ3: Is model training stability preserved under parallel orchestration?

Through controlled experimental analysis, this work seeks to provide practical insights into when pipeline parallelism is beneficial and when orchestration overhead negates its advantages, thereby supporting informed decision-making in ML pipeline orchestration design (Jain & Rajak, 2023; Rodrigues et al., 2025; Shwe & Aritsugi, 2024).

LITERATURE REVIEW

Workflow scheduling and parallel execution have been extensively studied in the context of Directed Acyclic Graph (DAG)–based applications. Prior research has primarily focused on the design and evaluation of scheduling algorithms aimed at optimizing makespan, resource utilization, and energy efficiency. Comprehensive surveys by (Verucchi et al., 2023) and (Eladgham et al., 2024) classify DAG scheduling techniques into priority-based methods, heuristic strategies, and schedulability analyses for parallel systems. These studies emphasize algorithmic properties and theoretical performance guarantees, often assuming stable execution environments with well-defined resource availability.

Subsequent research has extended classical DAG scheduling approaches by incorporating workload structure awareness and predictive mechanisms. (Albtoush et al., 2023) propose structure-aware scheduling methods for scientific workflows, while (Davami et al., 2021) introduce parallelism prediction techniques to improve scheduling decisions for multiple workflows under time constraints. Other studies evaluate makespan optimization and energy consumption in heterogeneous parallel systems, highlighting trade-offs between execution efficiency and resource usage (Chen, 2023; Hu et al., 2023; Stewart et al., 2023; Yang et al., 2023). However, these studies primarily emphasize algorithmic scheduling performance in distributed systems and do not empirically quantify orchestration overhead relative to workload size in single-node ML pipelines.

In parallel, empirical benchmarking studies have been conducted to assess workflow execution performance across various domains and infrastructures. (Krishnan et al., 2021) and (Agnihotri et al., 2025) demonstrate the importance of controlled benchmarking to isolate execution and orchestration effects on performance metrics such as makespan and throughput. (Kramer & Lu, 2025) directly compare bare-metal and orchestrated ML workflows, showing that orchestration layers can introduce significant overhead affecting execution time. However, many benchmarking studies focus on specific domains, such as bioinformatics workflows or stream processing systems,

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

and often target distributed, edge–fog–cloud, or high-performance computing environments (Čop et al., 2025; Elshamy et al., 2023; Shwe & Aritsugi, 2024; Sochat et al., 2025).

Recent studies have begun to highlight the challenges of workflow orchestration in resource-constrained environments. (Čop et al., 2025) report that AI workflow orchestration at the network edge exhibits performance characteristics that differ substantially from those observed in large-scale distributed systems. Similarly, research on task placement, bottleneck identification, and workflow optimization emphasizes that overhead and execution efficiency are highly context-dependent (K.M. & Shukla, 2025; Rodrigues et al., 2025; Tang et al., 2023). Therefore, there remains a lack of workload-aware experimental evidence explaining when orchestration parallelism becomes beneficial or detrimental in resource-constrained single-node environments. Unlike previous research that prioritizes scalability across multiple nodes, this study positions itself to fill this specific gap by explicitly analyzing the trade-off between orchestration costs and execution speed in local MLOps pipelines.

METHOD

Experimental Design and Problem Formulation

This study employs a controlled experimental design to evaluate the execution performance of Directed Acyclic Graph (DAG)–based machine learning pipelines in a single-node environment. The primary objective of this design is to isolate the performance impact of orchestration mechanisms by minimizing the influence of external and uncontrolled variables. It is important to note that the machine learning component described herein is not the primary subject of algorithmic evaluation but serves as a representative workload within a reproducible ML pipeline. Similar controlled approaches are widely adopted in workflow benchmarking and parallel system studies to ensure that observed performance differences originate from execution and orchestration behavior rather than from data variability or algorithmic complexity (Agnihotri et al., 2025; Krishnan et al., 2021; Verucchi et al., 2023).

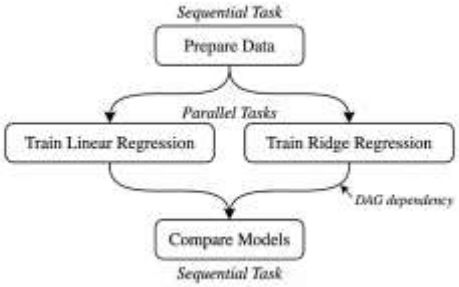

Fig. 1 DAG Structure machine learning pipeline

The machine learning pipeline analyzed in this study is modeled as a DAG consisting of four main tasks: data preparation (t_{prep}), Linear Regression model training (t_{LR}), Ridge Regression model training (t_{Ridge}), and model comparison $(t_{compare})$. As illustrated in Figure 1, the data preparation task must be completed before the two model training tasks can be executed. The training tasks have no direct dependency on each other and may therefore be executed in parallel, after which the pipeline proceeds to the model comparison stage. Representing pipelines as DAGs enables explicit modeling of task dependencies and potential parallelism, which is a common approach in workflow scheduling and performance analysis studies (Albtoush et al., 2023; Verucchi et al., 2023).

In the sequential execution scenario, all tasks are executed strictly according to the dependency order defined by the DAG structure, without any overlap between tasks. Consequently, the sequential makespan represents the total execution time required to complete all tasks in a single pipeline. The sequential makespan is formulated as:

$$\text{Makespan}_{seq} = C(t_{prep}) + C(t_{LR}) + C(t_{Ridge}) + C(t_{compare}) \qquad (1)$$

where $C(t)$ denotes the actual execution duration of task t, as recorded by the orchestration system. This sequential makespan serves as a baseline performance reference, representing the most conservative execution condition in which system resources are utilized by only one task at a time.

In the parallel execution scenario, the Linear Regression and Ridge Regression training tasks can be executed concurrently after the completion of the data preparation stage. Under this condition, the total pipeline execution time is no longer determined by the sum of all task durations, but rather by the longest execution path that constrains the overall completion time. This longest path, commonly referred to as the critical path, is defined as:

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

$$CP = \max_{p \in P} \sum_{t \in p} C(t) \qquad (2)$$

where P denotes the set of all possible execution paths in the DAG. The critical-path-based formulation represents a fundamental concept in makespan analysis for workflow scheduling and parallel systems, as it defines the lower bound of execution time that cannot be reduced even under maximal parallelism (Eladgham et al., 2024; Verucchi et al., 2023).

Based on the identified critical path, the parallel execution makespan for a single pipeline is formulated as:

$$\text{Makespan}_{par} = C(t_{prep}) + CP + C(t_{compare}) \qquad (3)$$

To quantify the relative performance impact of parallel execution compared to sequential execution, the speedup metric is employed. Speedup is defined as the ratio between the sequential makespan and the parallel makespan:

$$S = \frac{\text{Makespan}_{seq}}{\text{Makespan}_{par}} \qquad (4)$$

This metric is commonly used in the evaluation of parallel systems and DAG-based scheduling strategies to characterize acceleration or slowdown resulting from the application of parallelism (Eladgham et al., 2024; Stewart et al., 2023).

Pipeline Execution Model and Parallelism Semantics

The parallelism analyzed in this study is not limited to task-level execution within a single pipeline. Instead, the orchestration system is allowed to execute all tasks and pipeline instances that are in a ready state concurrently, as long as sufficient system resources remain available. This execution model enables both intra-pipeline parallelism, where independent tasks within the same pipeline are executed concurrently, and inter-pipeline parallelism, where multiple pipeline instances are executed simultaneously.

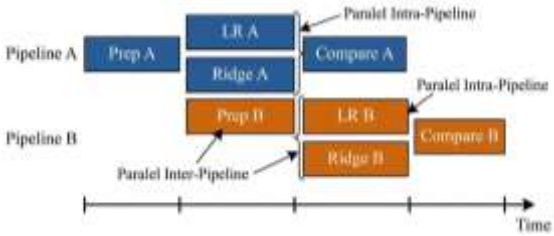

Fig. 2 Global Parallelism in Single-node Execution

To illustrate these execution semantics, Figure 2 depicts the execution timeline of two pipeline instances executed concurrently on a single-node system. Within each pipeline, the two model training tasks are executed in parallel after data preparation, representing intra-pipeline parallelism. At the same time, tasks belonging to different pipeline instances may overlap in execution, representing inter-pipeline parallelism. This execution behavior reflects a global parallelism model, in which the orchestrator adopts a work-conserving strategy by immediately assigning idle computational resources to other ready tasks or pipeline instances (Corodescu et al., 2021; Elshamy et al., 2023; Yasmin et al., 2025).

In the experimental context, each iteration consists of two pipeline instances executed concurrently on the same single-node environment. Accordingly, in addition to the per-pipeline makespan defined in Equation (3), this study defines the iteration makespan operationally as the time interval from the start of execution of the first pipeline instance to the completion of the last pipeline instance within a single iteration. This definition is used to represent the total completion time of the parallel workload at the experimental level and forms the basis for performance comparison in the Results section.

Experimental Environment, Monitoring, and Procedure

All experiments were conducted on a single Virtual Private Server (VPS) provided by DigitalOcean to eliminate variability caused by network latency, autoscaling, or dynamic resource allocation. The hardware

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

specifications included a server-class x86_64 architecture processor (Intel Xeon family) with 4 virtual CPUs (vCPUs), 8 GB of RAM, and 150 GB of SSD-based block storage. The operating system used was Ubuntu Linux 24.04 LTS with kernel version 6.8, and the experiments were executed using Python version 3.12. The machine learning pipeline was implemented using ZenML version 0.91.0, while pipeline orchestration and parallel execution were managed using Apache Airflow with the LocalExecutor in a container-based environment. This configuration reflects a common experimental and development setup in practical MLOps workflows, where orchestration mechanisms are often evaluated on single-node infrastructure prior to large-scale deployment (Kramer & Lu, 2025; Nelavelli & Augie, 2022; Steidl et al., 2023).

System-level resource utilization was monitored using Netdata, which continuously records CPU utilization and main memory (RAM) usage at a one-second temporal resolution during pipeline execution. Such fine-grained monitoring is commonly employed to support workflow performance analysis and to aid in the interpretation of makespan results and orchestration overhead (Elshamy et al., 2023; Krishnan et al., 2021). In addition to system-level metrics, machine learning performance metrics, including RMSE, MAE, and R^2, were recorded for each pipeline execution using MLflow. These metrics were collected to verify that different orchestration strategies and execution modes do not affect the stability of model training outcomes (Kramer & Lu, 2025; Steidl et al., 2023).

To simulate deterministic computational workloads, the execution durations of the model training tasks were explicitly controlled using time delays. The selection of delay values was based on the system's baseline orchestration overhead (approximately 30–60 seconds). Two scenarios were evaluated: (1) a light workload (30-second delay), representing a regime where task duration is comparable to or dominated by orchestration overhead, making the pipeline highly sensitive to scheduling costs; and (2) a heavy workload (240-second delay), representing a regime where task duration significantly exceeds the overhead, thereby rendering orchestration costs negligible relative to the total makespan. This approach is commonly adopted in workflow benchmarking studies to ensure that observed performance variations originate from execution and orchestration mechanisms rather than from algorithmic or data-related variability (Agnihotri et al., 2025; Krishnan et al., 2021; Sochat et al., 2025).

The dataset used in the experiments consists of daily NVIDIA (NVDA) stock price time-series data in tabular form, limited to the most recent 1,000 records. The dataset includes features such as opening price, closing price, and trading volume, along with derived features in the form of lagging values and statistical volatility measures. Model training was performed using Linear Regression and Ridge Regression with hyperparameter optimization via Grid Search.

Performance evaluation was conducted by comparing measured execution results with the theoretical formulations defined in Equations (1)–(4). The sequential makespan, computed as the sum of task execution durations, serves as the baseline reference. For parallel execution, the critical-path-based formulation provides a theoretical lower bound for completion time, while the measured iteration makespan captures the practical performance of concurrent pipeline execution. CPU and memory utilization metrics were analyzed descriptively to support the interpretation of orchestration overhead, and machine learning evaluation metrics were compared across execution scenarios to confirm the stability of model training outcomes rather than to assess improvements in predictive accuracy.

Data analysis was performed using both descriptive and inferential approaches. Performance metrics are reported as mean ± standard deviation (SD). To assess the statistical significance of differences between sequential and parallel execution makespan, a Mann–Whitney U test was applied, considering the small sample size ($n = 10$). Statistical significance was determined using a threshold of $p < 0.05$. In addition, effect size was quantified using Cohen's d to evaluate the magnitude of the observed performance differences.

RESULT

This section presents the experimental results of machine learning pipeline execution performance in a single-node environment. The evaluation focuses on pipeline execution makespan as the primary performance indicator, complemented by an analysis of system resource utilization patterns, including CPU and memory usage, to characterize execution behavior under different orchestration strategies. In addition, machine learning performance metrics are reported to verify the stability and determinism of model training outcomes across all execution scenarios. All experiments were conducted under identical hardware and software configurations, and the same pipeline structure, dataset, and execution parameters were maintained throughout the evaluation to ensure that the reported results are directly comparable.

The experimental evaluation covers four scenarios derived from combinations of orchestration strategy and simulated workload duration, namely execution using the Local Orchestrator and the Airflow Orchestrator under light and heavy workload conditions. For each scenario, performance measurements are reported across ten experimental iterations, where each iteration consists of two pipeline instances executed on the same single-node system. Detailed execution timestamps are recorded for the start and completion of each pipeline instance, and these timestamps are used to compute the iteration makespan, which represents the total completion time of the

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

aggregated workload within a single iteration. Alongside makespan measurements, CPU and memory utilization are monitored continuously during pipeline execution, and machine learning evaluation metrics are collected for every pipeline run to ensure that differences in orchestration strategy and execution mode do not affect model accuracy or consistency.

Local Orchestrator – Light Workload

This scenario evaluates pipeline execution using the Local Orchestrator under a light workload, which is simulated by applying a fixed execution delay during the model training stage. Each experimental iteration consists of two pipeline instances executed sequentially on the same single-node system. All results are obtained from ten iterations.

The iteration makespan is defined as the time interval between the start of execution of the first pipeline instance and the completion of execution of the second pipeline instance within the same iteration.

Table 1. Makespan per Iteration for Local Orchestrator under Light Workload

Iterasi	Start Pipeline A	End Pipeline A	Start Pipeline B	End Pipeline B	Makespan Iterasi (s)
1	2:12:45	2:14:08	2:14:12	2:15:33	168
2	2:15:40	2:17:02	2:17:06	2:18:28	168
3	2:18:35	2:19:54	2:19:57	2:21:22	167
4	2:21:29	2:22:50	2:22:54	2:24:17	168
5	2:24:25	2:25:47	2:25:51	2:27:13	168
6	2:27:21	2:28:41	2:28:44	2:30:05	164
7	2:30:14	2:31:38	2:31:42	2:33:03	169
8	2:33:10	2:34:31	2:34:34	2:35:55	165
9	2:36:02	2:37:23	2:37:27	2:38:49	167
10	2:38:56	2:40:19	2:40:23	2:41:46	170

Table 1 presents the makespan per iteration for pipeline execution using the Local Orchestrator under the light workload scenario. The observed makespan values range from 164 to 170 seconds, with an average value of 167.4 seconds. The variation across iterations is relatively small, indicating stable and deterministic execution behavior across repeated runs. Minor fluctuations in makespan are attributable to small differences in task execution timing and orchestration overhead, rather than to changes in pipeline structure or workload characteristics. Since the two pipeline instances are executed sequentially without any execution overlap, the measured makespan represents the cumulative execution time required to complete both pipelines within a single iteration, reflecting a conservative execution model in which system resources are utilized by only one pipeline at a time.

Fig. 3 CPU utilization profile during sequential pipeline execution using Local Orchestrator under light workload

System resource utilization during pipeline execution was monitored using Netdata to observe CPU and memory usage behavior throughout the experimental runs. Figure 3 illustrates the CPU utilization profile during sequential pipeline execution using the Local Orchestrator under the light workload scenario. The CPU utilization pattern exhibits periodic increases corresponding to the execution phases of the model training tasks, followed by a return

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

to idle conditions after task completion. These recurring fluctuations indicate that computational activity is concentrated within clearly defined execution intervals, while extended idle periods occur between task executions. CPU idle values remain high throughout the experiment, reflecting the absence of parallel execution across tasks and pipeline instances and indicating that available processing resources are not continuously engaged during pipeline execution. Overall, the observed CPU utilization pattern is consistent across iterations and shows no irregular spikes or sustained load that would suggest uncontrolled resource contention or instability at the system level.

Fig. 4 System memory utilization during sequential pipeline execution using Local Orchestrator under light workload

Figure 4 shows the system memory utilization during sequential pipeline execution using the Local Orchestrator under the light workload scenario. Used memory fluctuates within a narrow range of approximately 3.45–3.65 GiB throughout the execution period, indicating consistent memory consumption across pipeline stages. At the same time, free and cached memory remain continuously available, reflecting stable memory allocation behavior at the operating system level. The absence of sharp spikes, sustained increases in used memory, or progressive growth over time suggests that pipeline execution does not introduce memory pressure or excessive dynamic allocation. Overall, the observed memory utilization pattern remains consistent across iterations and indicates that sequential pipeline execution under the light workload scenario is accommodated well within the available memory capacity without inducing instability.

Table 2. Summary of Machine Learning Metrics Across 10 Iterations (Light Workload, Local Orchestrator)

Model	Metric	Min	Max	Mean	Std
Linear Regression	RMSE	2.754	2.754	2.754	0.0000
Linear Regression	MAE	1.945	1.945	1.945	0.0000
Linear Regression	R^2	0.988	0.988	0.988	0.0000
Ridge Regression	RMSE	2.741	2.741	2.741	0.0000
Ridge Regression	MAE	1.923	1.923	1.923	0.0000
Ridge Regression	R^2	0.988	0.988	0.988	0.0000

Machine learning performance metrics were recorded for each pipeline execution using MLflow. Table 2 summarizes the machine learning metrics across ten iterations for the Local Orchestrator under the light workload scenario. The results show that the Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and coefficient of determination (R^2) for both Linear Regression and Ridge Regression models are identical across all iterations, with a standard deviation of zero. These results indicate fully deterministic model training behavior under sequential execution. The consistency of these metrics across repeated runs confirms that variations in execution order and system-level scheduling do not influence the numerical outcomes of model training in this scenario.

Airflow Orchestrator – Light Workload

This scenario evaluates pipeline execution using the Airflow Orchestrator under the same light workload conditions. In this configuration, parallel execution across tasks and pipeline instances is enabled, while all other

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

experimental parameters, including pipeline structure, workload characteristics, and execution environment, remain unchanged to ensure comparability with the sequential baseline.

In each iteration, two pipeline instances are executed concurrently on the same single-node system. The iteration makespan is defined as the time interval between the earliest pipeline start time and the latest pipeline completion time. This definition is used to capture the total completion time of the aggregated workload under parallel execution and serves as the primary performance measure for comparing parallel and sequential orchestration behavior in this scenario.

Table 3. Makespan per Iteration for Airflow Orchestrator under Light Workload

Iterasi	Start Pipeline A	End Pipeline A	Start Pipeline B	End Pipeline B	Makespan Iterasi (s)
1	1:11:04	1:14:28	1:11:19	1:14:46	222
2	1:15:26	1:19:04	1:15:40	1:19:17	231
3	1:20:02	1:23:47	1:20:18	1:23:53	231
4	1:24:39	1:28:21	1:24:54	1:28:28	229
5	1:29:10	1:33:05	1:29:26	1:33:23	253
6	1:33:57	1:37:54	1:34:13	1:37:57	240
7	1:38:34	1:42:23	1:38:48	1:42:31	237
8	1:43:04	1:46:51	1:43:20	1:46:52	228
9	1:47:28	1:51:14	1:47:43	1:51:16	228
10	1:51:51	1:55:55	1:52:05	1:55:59	248

Table 3 presents the makespan per iteration for pipeline execution using the Airflow Orchestrator under the light workload scenario. The observed makespan values range from 222 to 253 seconds, with an average value of 234.7 seconds. The overlapping execution periods of the two pipeline instances indicate that global parallelism is applied during execution. Despite the application of parallel execution, the measured makespan is higher than that observed in the sequential baseline under light workload conditions.

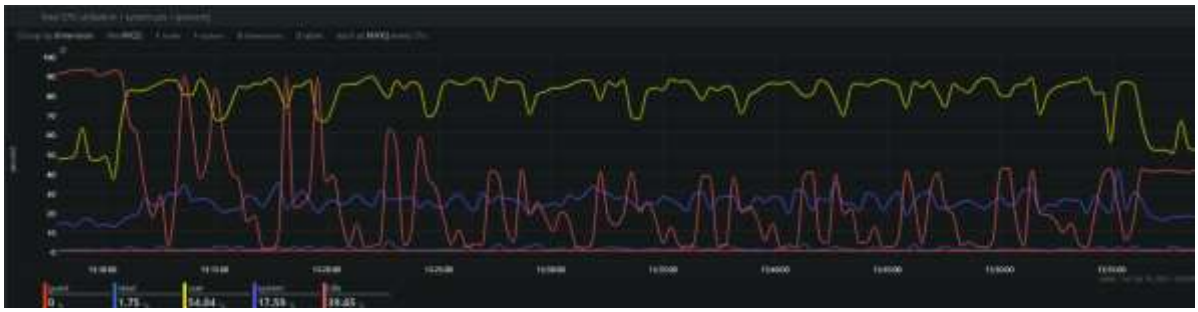

Fig. 5 CPU utilization profile during parallel pipeline execution using Airflow Orchestrator under light workload

System resource utilization during pipeline execution with the Airflow Orchestrator is shown in the following figures. Figure 5 illustrates the CPU utilization profile during parallel pipeline execution using the Airflow Orchestrator under the light workload scenario. CPU utilization increases significantly compared to sequential execution, with recurring peaks corresponding to the parallel execution of model training tasks. CPU idle values decrease substantially, indicating more active utilization of available processing resources.

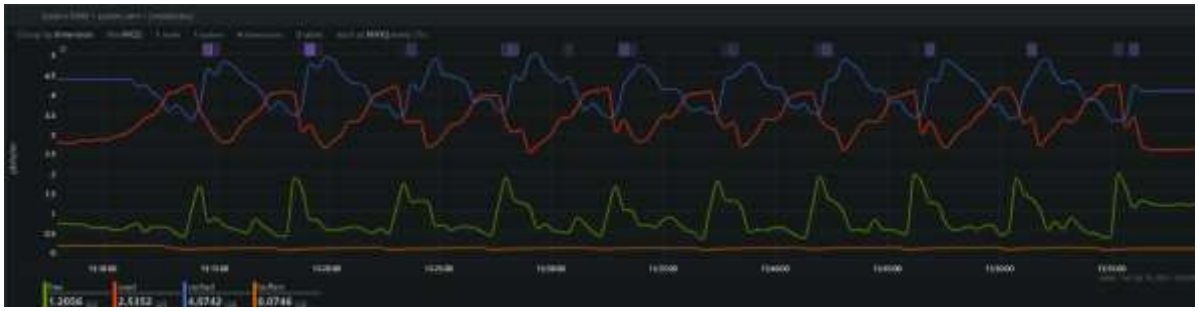

Fig. 6 System memory utilization during parallel pipeline execution using Airflow Orchestrator under light workload

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Figure 6 shows the system memory utilization during parallel pipeline execution using the Airflow Orchestrator under the light workload scenario. Used memory exhibits cyclical but controlled fluctuations in accordance with parallel task execution phases, while cached and free memory remain available throughout the experiment. No memory contention or instability is observed.

Table 4. Summary of Machine Learning Metrics Across 10 Iterations (Light Workload, Airflow Orchestrator)

Model	Metric	Min	Max	Mean	Std
Linear Regression	RMSE	2.754	2.754	2.754	0.0000
Linear Regression	MAE	1.945	1.945	1.945	0.0000
Linear Regression	R^2	0.988	0.988	0.988	0.0000
Ridge Regression	RMSE	2.741	2.741	2.741	0.0000
Ridge Regression	MAE	1.923	1.923	1.923	0.0000
Ridge Regression	R^2	0.988	0.988	0.988	0.0000

Table 4 summarizes the machine learning metrics across ten iterations for the Airflow Orchestrator under the light workload scenario. All metrics remain identical across iterations with zero variance, indicating that enabling parallel execution does not affect the stability or determinism of model training outcomes.

Local Orchestrator – Heavy Workload

This scenario evaluates pipeline execution using the Local Orchestrator under a heavy workload, simulated by applying a longer fixed execution delay during model training.

Table 5. Makespan per Iteration for Local Orchestrator under Heavy Workload

Iterasi	Start Pipeline A	End Pipeline A	Start Pipeline B	End Pipeline B	Makespan Iterasi (s)
1	9:19:35	9:27:59	9:28:02	9:36:23	1008
2	9:36:30	9:44:51	9:44:54	9:53:17	1007
3	9:53:24	10:01:53	10:01:56	10:10:21	1017
4	10:10:29	10:18:52	10:18:56	10:27:20	1011
5	10:27:28	10:35:50	10:35:55	10:44:18	1010
6	10:44:25	10:52:47	10:52:50	11:01:15	1010
7	11:01:23	11:09:50	11:09:53	11:18:17	1014
8	11:18:24	11:26:47	11:26:50	11:35:22	1018
9	11:35:30	11:43:55	11:43:58	11:52:21	1011
10	11:52:28	12:00:52	12:00:56	12:09:28	1020

Table 5 presents the makespan per iteration for pipeline execution using the Local Orchestrator under the heavy workload scenario. The observed makespan values range from 1007 to 1020 seconds, with an average value of 1012.6 seconds. The small variation across iterations indicates stable and deterministic execution behavior. As in the light workload scenario, the pipeline instances are executed sequentially without overlap, and the measured makespan reflects the cumulative execution time of both pipelines under prolonged computation.

Fig. 7 CPU utilization profile during sequential pipeline execution using Local Orchestrator under heavy workload

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

System resource utilization during execution under the heavy workload scenario was also monitored. Figure 7 illustrates the CPU utilization profile during sequential pipeline execution using the Local Orchestrator under the heavy workload scenario. CPU utilization remains higher and more sustained compared to the light workload scenario, particularly during the prolonged model training phases, while still exhibiting idle periods due to the absence of parallel execution.

Fig. 8 System memory utilization during sequential pipeline execution using Local Orchestrator under heavy workload

Figure 8 shows the system memory utilization during sequential pipeline execution using the Local Orchestrator under the heavy workload scenario. Memory usage remains stable throughout the experiment, with no signs of memory contention, excessive allocation, or abnormal behavior.

Table 6. Summary of Machine Learning Metrics Across 10 Iterations (Heavy Workload, Local Orchestrator)

Model	Metric	Min	Max	Mean	Std
Linear Regression	RMSE	2.754	2.754	2.754	0.0000
Linear Regression	MAE	1.945	1.945	1.945	0.0000
Linear Regression	R^2	0.988	0.988	0.988	0.0000
Ridge Regression	RMSE	2.741	2.741	2.741	0.0000
Ridge Regression	MAE	1.923	1.923	1.923	0.0000
Ridge Regression	R^2	0.988	0.988	0.988	0.0000

Table 6 summarizes the machine learning metrics across ten iterations for the Local Orchestrator under the heavy workload scenario. All recorded metrics remain identical across iterations, confirming deterministic model behavior under sequential execution with heavy workload.

Airflow Orchestrator – Heavy Workload

This scenario evaluates pipeline execution using the Airflow Orchestrator under the heavy workload condition, with parallel execution enabled.

Table 7. Makespan per Iteration for Airflow Orchestrator under Heavy Workload

Iterasi	Start Pipeline A	End Pipeline A	Start Pipeline B	End Pipeline B	Makespan Iterasi (s)
1	12:47:53	12:56:46	12:48:16	12:56:48	535
2	12:57:45	1:06:04	12:58:04	1:06:43	538
3	1:07:41	1:16:44	1:08:02	1:16:44	543
4	1:17:44	1:26:40	1:18:04	1:26:48	544
5	1:27:48	1:36:49	1:28:10	1:36:50	542
6	1:37:53	1:46:42	1:38:16	1:46:51	538
7	1:47:41	1:56:18	1:48:01	1:56:31	530
8	1:57:28	2:05:43	1:57:48	2:06:01	513
9	2:06:52	2:15:26	2:07:14	2:15:28	516
10	2:16:27	2:24:54	2:16:50	2:25:09	522

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Table 7 presents the makespan per iteration for pipeline execution using the Airflow Orchestrator under the heavy workload scenario. The observed makespan values range from 513 to 544 seconds, with an average value of 532.1 seconds. The relatively narrow variation range indicates stable execution despite the intensive computational workload. The measured makespan reflects the actual completion time of two pipeline instances executed concurrently.

Fig. 9 CPU utilization profile during parallel pipeline execution using Airflow Orchestrator under heavy workload

Figure 9 illustrates the CPU utilization profile during parallel pipeline execution using the Airflow Orchestrator under the heavy workload scenario. CPU utilization remains consistently high throughout the experiment, reflecting sustained parallel execution of model training tasks at both intra-pipeline and inter-pipeline levels. The utilization pattern exhibits recurring fluctuations that correspond to the execution phases of concurrent training tasks, while idle CPU values remain relatively low for most of the execution period. This pattern indicates that available processing resources are continuously engaged during pipeline execution without prolonged idle intervals or abrupt utilization drops.

Fig. 10 System memory utilization during parallel pipeline execution using Airflow Orchestrator under heavy workload

Figure 10 shows the system memory utilization during parallel pipeline execution using the Airflow Orchestrator under the heavy workload scenario. Memory usage remains within a controlled range without evidence of resource exhaustion, instability, or abnormal allocation behavior.

Table 8. Summary of Machine Learning Metrics Across 10 Iterations (Heavy Workload, Airflow Orchestrator)

Model	Metric	Min	Max	Mean	Std
Linear Regression	RMSE	2.754	2.754	2.754	0.0000
Linear Regression	MAE	1.945	1.945	1.945	0.0000
Linear Regression	R^2	0.988	0.988	0.988	0.0000
Ridge Regression	RMSE	2.741	2.741	2.741	0.0000
Ridge Regression	MAE	1.923	1.923	1.923	0.0000
Ridge Regression	R^2	0.988	0.988	0.988	0.0000

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Table 8 summarizes the machine learning metrics across ten iterations for the Airflow Orchestrator under the heavy workload scenario. All machine learning metrics exhibit identical values across iterations with zero variance, confirming that parallel execution under heavy workload conditions does not affect model training stability.

Statistical Performance

To quantify the performance differences observed in the iteration data and address the variability across runs, a statistical summary is presented in Table 9.

Table 9. Statistical Summary of Makespan (Seconds) and Speedup Metrics

Workload Scenario	Execution Mode	Mean Makespan (s)	Std. Dev (SD)	Speedup	p-value
Light (30s)	Sequential	167.40	± 1.84	-	-
	Parallel	234.70	± 10.31	0.71×	< 0.001
Heavy (240s)	Sequential	1,012.60	± 4.50	-	-
	Parallel	532.10	± 10.58	1.90×	< 0.001

Table 9 confirms that pipeline performance is strongly dependent on workload duration, with statistically significant differences between sequential and parallel execution confirmed by the Mann–Whitney U test ($p < 0.001$).

Under the light workload scenario (30-second tasks), the system exhibits a negative speedup of 0.71×. The measured parallel makespan, with a mean value of 234.70 seconds, consistently exceeds the sequential baseline, which has a mean makespan of 167.40 seconds. This result indicates that fixed orchestration overhead, including DAG parsing, task scheduling, and container initialization, is statistically significant relative to the short task duration, thereby negating the potential benefits of concurrent execution.

In contrast, under the heavy workload scenario (240-second tasks), the system achieves a substantial positive speedup of 1.90×. The parallel execution yields a mean makespan of 532.10 seconds, which is nearly half of the sequential execution time with a mean of 1,012.60 seconds. This result demonstrates that as task execution duration increases, the relative impact of orchestration overhead becomes negligible, allowing parallelism to effectively reduce overall completion time.

The visual comparison of these results, including the variability indicated by the standard deviation, is presented in Figure 11.

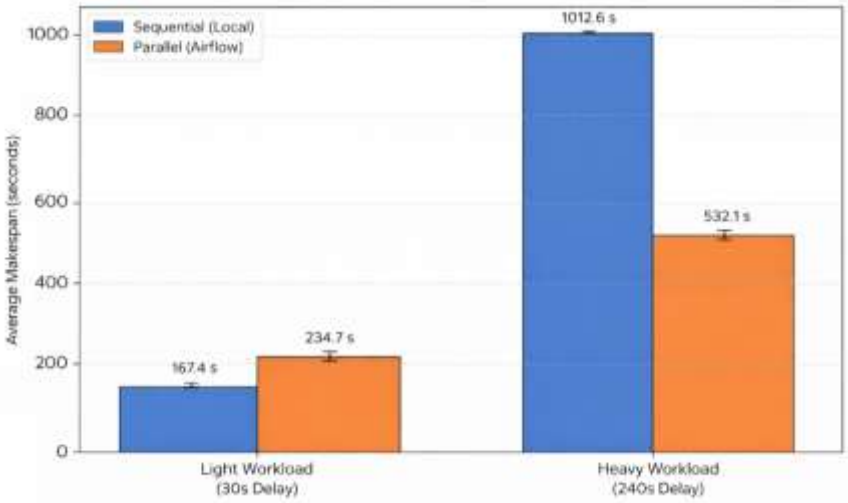

Fig. 11 Comparison of average makespan showing the shift from negative to positive speedup across workloads.

Error bars represent standard deviation.

Observed Workload Threshold Based strictly on the experimental data, a crossover point or workload boundary is observed. The results indicate that parallel execution only yields performance benefits (Speedup > 1.0) when the task duration significantly exceeds the system's baseline overhead (estimated at 30–60 seconds). Below this threshold, the measured makespan increases due to orchestration costs; above it, the computational efficiency of parallel execution prevails.

* Rama Aria Megantara

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 10, Number 1, January 2026
DOI : https://doi.org/10.33395/sinkron.v10i1.15788
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Resource Utilization & ML Metrics Summary The CPU profiles presented earlier (Figs. 3–10) corroborate these findings: heavy workloads show sustained parallel CPU usage, whereas light workloads show fragmentation. Additionally, the supplementary data confirms that Machine Learning metrics (RMSE, MAE, R^2) maintained zero variance across all 40 experimental runs, ensuring that the speedup gains in the heavy scenario did not come at the cost of model accuracy.

DISCUSSIONS

This study provides an empirical investigation into the performance impact of machine learning pipeline orchestration in a single-node environment, with particular emphasis on makespan behavior, orchestration overhead, system resource utilization, and the stability of model training outcomes. The discussion interprets the experimental results by relating them to concepts from Directed Acyclic Graph (DAG) scheduling theory, workflow orchestration, and Machine Learning Operations (MLOps) practices reported in prior studies (Kramer & Lu, 2025; Nelavelli & Augie, 2022; Steidl et al., 2023).

Under the light workload scenario, the experimental results show that applying parallel orchestration using Apache Airflow does not lead to an improvement in execution performance when compared to sequential execution using the Local Orchestrator. Instead, the measured iteration makespan increases substantially, indicating inefficiency rather than acceleration. This behavior suggests that, for workloads with relatively short computation durations, the system operates in an overhead-bound regime, where orchestration overhead dominates the total execution time. Similar bottleneck-oriented analyses have been reported in prior studies, where system management costs outweigh the benefits of parallelism in small-scale workflows, even when the DAG structure theoretically supports concurrent execution (Tang et al., 2023).

The observed inefficiency under light workloads can be attributed to the imbalance between computation time and orchestration management costs. When task execution durations are short, the overhead introduced by orchestrator-level operations specifically DAG parsing, scheduler heartbeat intervals, inter-process communication, and container initialization becomes comparable to, or even exceeds, the core computation time. As a result, the potential benefits of executing multiple pipelines concurrently are insufficient to offset the additional orchestration costs. This finding is consistent with previous reports indicating that, for lightweight workflows, the advantages of parallel execution are often negated by orchestration overhead (Elshamy et al., 2023; Theusch & Heisterkamp, 2024).

The effectiveness of transitioning from sequential to parallel execution is further reflected in the observed speedup value, which is below unity for the light workload scenario. A speedup value less than one indicates that parallel orchestration results in execution slowdown rather than acceleration. It should be noted that, although the theoretical parallel makespan is defined based on the critical path of a single pipeline instance, the experimental evaluation in this study analyzes makespan at the iteration level, capturing the aggregate effects of inter-pipeline parallelism and orchestration overhead simultaneously. This aggregation explains why parallel execution becomes inefficient for short-running workloads, as also reported in studies on machine learning pipeline optimization based on task placement strategies (Rodrigues et al., 2025).

In contrast, under the heavy workload scenario, the application of parallel orchestration using Apache Airflow leads to a substantial reduction in iteration makespan compared to sequential execution. When computation durations are significantly longer, the costs associated with orchestration management become relatively small compared to the total execution time. In this condition, the system transitions to a compute-bound regime, where inter-pipeline and intra-pipeline parallelism can be exploited effectively, resulting in execution times that approach the duration of the longest pipeline rather than the cumulative execution time of multiple pipelines.

This behavior aligns with fundamental principles of DAG scheduling theory in parallel systems, where the achievable makespan approaches the theoretical lower bound defined by the critical path length rather than the cumulative sum of all task durations (Chen, 2023; Eladgham et al., 2024). Furthermore, schedulability analyses in parallel and multicore systems indicate that, even when DAG structures support parallelism, the achievable performance gains are bounded by architectural characteristics and execution critical paths (Yang et al., 2023). The observed speedup greater than one under the heavy workload scenario confirms that parallel orchestration becomes advantageous once computation costs dominate orchestration overhead.

These findings support the existence of a workload threshold beyond which parallel execution yields tangible performance benefits. For computation-intensive workloads, orchestration frameworks are able to utilize available system resources more effectively, leading to significant reductions in total completion time. Similar observations have been reported in studies on workflow orchestration in cloud and edge–cloud environments, which emphasize that parallelism becomes effective when computation dominates management overhead (Jain & Rajak, 2023; K.M. & Shukla, 2025; Shwe & Aritsugi, 2024).

Another important outcome of this study is the complete stability of machine learning evaluation metrics across all execution scenarios. The observed consistency of RMSE, MAE, and R^2 values across repeated iterations indicates that model training outcomes are fully deterministic under the experimental conditions employed. This

* Rama Aria Megantara

e-ISSN : 2541-2019
p-ISSN : 2541-044X

confirms that changes in orchestration strategy and execution mode do not influence the numerical results of model training as long as the data, parameters, and training configurations remain unchanged.

The stability of machine learning metrics has important methodological implications. It validates the experimental design by confirming that the orchestration strategy influences only the temporal aspects of execution (i.e., makespan and resource timing), not the functional correctness or determinism of the model training process. This clear separation between system performance evaluation and model quality assessment is consistent with benchmarking practices emphasized in MLOps and machine learning pipeline research (Kramer & Lu, 2025; Nelavelli & Augie, 2022; Steidl et al., 2023).

Scientific Contributions and Theoretical Implications

Beyond the empirical performance data, this study offers several key scientific contributions to the field of MLOps pipeline engineering and workflow scheduling:

1. Empirical Validation of the Orchestration "Break-Even Point": This research provides concrete empirical evidence for a theoretical workload threshold in single-node orchestration. We demonstrate that the transition from negative speedup to positive speedup is characterized by a distinct "break-even point" (estimated between 30–60 seconds in this environment), below which the complexity of a full-fledged orchestrator is theoretically unjustified due to overhead penalties.
2. Formalization of Overhead Dominance in ML Workflows: The study formalizes the concept of "overhead dominance" specifically for short-duration ML workflows. It contributes to the theoretical understanding that in lightweight task regimes, the orchestration layer itself acts as the primary critical path constraint, rendering DAG parallelism ineffective.
3. Critical-Path vs. Overhead Cost Trade-off Model: The findings refine existing DAG scheduling theories by integrating a tangible cost model for orchestration overhead in resource-constrained environments. This contradicts idealistic assumptions in some scheduling literature that ignore initialization costs, emphasizing the need for overhead-aware scheduling algorithms in edge or single-server MLOps deployments [CITE: Jain & Rajak, 2023; Shwe & Aritsugi, 2024].

Threats to Validity

While the findings are statistically robust within the experimental scope, several limitations regarding their generalizability must be acknowledged:

1. Single-Node Constraint: The experiments were conducted in a single-node VPS environment with limited resources (4 vCPUs). The observed overhead behavior and threshold values may differ significantly in distributed, multi-node cluster environments (e.g., Kubernetes), where network latency, distributed state management, and data locality introduce additional layers of complexity.
2. Artificial Workload Simulation: Workload duration was controlled using fixed delays to ensure precise experimental conditions. While necessary for isolating variables, this approach may not fully capture the dynamic CPU and memory contention patterns of real-world, intensive mathematical computations during model training.
3. Limited Pipeline Structure: The evaluation employed a relatively simple parallel-branch DAG structure. The impact of orchestration overhead might be more pronounced or behave differently in highly complex DAGs characterized by numerous short, interdependent tasks and high fan-out/fan-in ratios.

From a practical perspective, the results of this study highlight that machine learning pipeline orchestration should not follow a one-size-fits-all approach. For development, prototyping, and workloads with short execution durations, sequential orchestration using lightweight mechanisms may provide more predictable and efficient performance by avoiding unnecessary overhead. Conversely, for computation-intensive pipelines, orchestration frameworks that support global parallelism are more suitable, as they can significantly reduce execution time by maximizing resource utilization.

Overall, these findings suggest that effective machine learning pipeline orchestration requires a workload-aware and adaptive approach. Execution strategies should be selected based on workload characteristics, DAG structure, and available infrastructure resources. Such adaptive orchestration strategies are increasingly emphasized in the workflow orchestration and MLOps literature, where systems are expected to adjust execution behavior according to the context of the workload being processed (Karthik et al., 2023; Rodrigues et al., 2025; Singh et al., 2020).

CONCLUSION

This study empirically investigated the performance trade-offs between sequential (Local) and parallel (Apache Airflow) orchestration strategies for machine learning pipelines in a resource-constrained, single-node environment. The experimental results reveal a critical dependency between orchestration efficiency and workload duration. While parallel execution achieved a significant speedup of approximately 1.90× under heavy, computation-intensive workloads, it resulted in a marked performance degradation (0.71× speedup) under light

* Rama Aria Megantara

workloads due to the dominance of orchestration overhead. Furthermore, the complete stability of machine learning evaluation metrics across all scenarios confirms that orchestration choices affect execution timing without compromising the determinism of model training outcomes.

Theoretically, these findings validate the existence of a distinct workload threshold or "break-even point" in single-node DAG scheduling, demonstrating that below this threshold, the system operates in an overhead-bound regime where fixed costs negate concurrency benefits. As suggested by the empirical evidence, this study provides strong support for workload-aware orchestration design and demonstrates that pipeline parallelism is not universally beneficial in single-node ML environments. The primary scientific contribution lies in formalizing the trade-off between critical-path reduction and orchestration overhead cost, underscoring the necessity for adaptive scheduling strategies in MLOps.

Regarding directions for future research, it is important to note that the current study utilized controlled artificial delays to simulate workload durations, as the primary focus was on isolating orchestration mechanics rather than the complexity of the machine learning tasks themselves. Therefore, future work should extend this evaluation by incorporating highly complex, real-world machine learning workloads, such as intensive deep learning training tasks involving large datasets and GPU utilization. Investigating how orchestration overhead interacts with the dynamic resource contention characteristic of these complex models will provide further insights into optimizing MLOps pipelines in production environments.

REFERENCES

Agnihotri, P., Koldehofe, B., Heinrich, R., Binnig, C., & Luthra, M. (2025). PDSP-BENCH: A Benchmarking System for Parallel and Distributed Stream Processing. *Proceedings of the ACM SIGMOD/PODS International Conference on Management of Data*. https://doi.org/10.1145/3722212.3725100

Albtoush, A., Yunus, F., Almi'ani, K., & Noor, N. M. M. (2023). Structure-Aware Scheduling Methods for Scientific Workflows in Cloud. *Applied Sciences (Switzerland)*, *13*(3). https://doi.org/10.3390/app13031980

Chen, C. Y. (2023). Scheduling coflows for minimizing the total weighted completion time in heterogeneous parallel networks. *Journal of Parallel and Distributed Computing*, *182*. https://doi.org/10.1016/j.jpdc.2023.104752

Čop, A., Bertalanič, B., & Fortuna, C. (2025). An overview and solution for democratizing AI workflows at the network edge. *Journal of Network and Computer Applications*, *239*. https://doi.org/10.1016/j.jnca.2025.104180

Corodescu, A. A., Nikolov, N., Khan, A. Q., Soylu, A., Matskin, M., Payberah, A. H., & Roman, D. (2021). Big data workflows: Locality-aware orchestration using software containers. *Sensors*, *21*(24). https://doi.org/10.3390/s21248212

Davami, F., Adabi, S., Rezaee, A., & Rahmani, A. M. (2021). Distributed scheduling method for multiple workflows with parallelism prediction and DAG prioritizing for time constrained cloud applications. *Computer Networks*, *201*. https://doi.org/10.1016/j.comnet.2021.108560

Eladgham, A. A., Ziedan, N. I., & Ziedan, I. (2024). Scheduling Algorithms in Parallel Processing: A Survey. *The Egyptian International Journal of Engineering Sciences and Technology*, *0*(0), 0–0. https://doi.org/10.21608/eijest.2024.325850.1294

Elshamy, A., Alquraan, A., & Al-Kiswany, S. (2023). *A Study of Orchestration Approaches for Scientific Workflows in Serverless Computing*. 34–40. https://doi.org/10.1145/3592533.3592809

Hu, B., Yang, X., & Zhao, M. (2023). Online energy-efficient scheduling of DAG tasks on heterogeneous embedded platforms. *Journal of Systems Architecture*, *140*. https://doi.org/10.1016/j.sysarc.2023.102894

Jain, A., & Rajak, R. (2023). A systematic review of workflow scheduling techniques in a fog environment. In *International Journal of Experimental Research and Review* (Vol. 30, pp. 100–108). International Academic Publishing House (IAPH). https://doi.org/10.52756/ijerr.2023.v30.011

Karthik, G. M., Gupta, A., Rajeshgupta, S., Jha, A., Sivasangari, A., & Mishra, B. P. (2023). Efficient Task Scheduling in Cloud Environment Based On Dynamic Priority and Optimized Technique. *2023 International Conference on Artificial Intelligence and Smart Communication, AISC 2023*, 1124–1129. https://doi.org/10.1109/AISC56616.2023.10085447

K.M., U., & Shukla, S. (2025). Energy and performance-aware workflow scheduler using dynamic virtual network resource optimization under edge-cloud platform. *Computers and Electrical Engineering*, *123*. https://doi.org/10.1016/j.compeleceng.2025.110085

Kramer, J., & Lu, T. (2025). A Reproducible Framework for Benchmarking Machine Learning Operations (MLOps) Infrastructures: Comparing Bare-Metal and Orchestrated Machine Learning Workflows. *Cureus Journal of Computer Science*. https://doi.org/10.7759/s44389-025-08693-x

Krishnan, V., Utiramerur, S., Ng, Z., Datta, S., Snyder, M. P., & Ashley, E. A. (2021). Benchmarking workflows to assess performance and suitability of germline variant calling pipelines in clinical diagnostic assays. *BMC Bioinformatics*, *22*(1). https://doi.org/10.1186/s12859-020-03934-3

* Rama Aria Megantara

Nelavelli, S., & Augie, M. A. (2022). MLOps in the Enterprise Cloud: Orchestrating Machine Learning Pipelines with Kubernetes. *Sarcouncil Journal of Engineering and Computer Sciences*, *04*(09). https://doi.org/10.5281/zenodo.17214373

Rodrigues, P., Corona, J., Antunes, M., & Aguiar, R. L. (2025). Optimising ML Pipeline Execution via Smart Task Placement. *Electronics (Switzerland)*, *14*(13). https://doi.org/10.3390/electronics14132555

Shwe, T., & Aritsugi, M. (2024). Optimizing Data Processing: A Comparative Study of Big Data Platforms in Edge, Fog, and Cloud Layers. *Applied Sciences (Switzerland)*, *14*(1). https://doi.org/10.3390/app14010452

Singh, A., Purawat, S., Rao, A., & Altintas, I. (2020). Modular performance prediction for scientific workflows using Machine Learning. *Future Generation Computer Systems*, *114*, 1–14. https://doi.org/10.1016/j.future.2020.04.048

Sochat, V., Pottier, L., & Milroy, D. (2025). *State Machine Orchestration of an HPC Workflow in Cloud*. 2293–2304. https://doi.org/10.1145/3731599.3767583

Steidl, M., Felderer, M., & Ramler, R. (2023). The pipeline for the continuous development of artificial intelligence models – Current state of research and practice. *Journal of Systems and Software*, *199*, 111615. https://doi.org/10.1016/j.jss.2023.111615

Stewart, R., Raith, A., & Sinnen, O. (2023). Optimising makespan and energy consumption in task scheduling for parallel systems. *Computers and Operations Research*, *154*. https://doi.org/10.1016/j.cor.2023.106212

Tang, J., Liu, Y., Lin, K. yi, & Li, L. (2023). Process bottlenecks identification and its root cause analysis using fusion-based clustering and knowledge graph. *Advanced Engineering Informatics*, *55*. https://doi.org/10.1016/j.aei.2022.101862

Theusch, F., & Heisterkamp, P. (2024). *Comparative Analysis of Open-Source ML Pipeline Orchestration Platforms* [Bachelor's Thesis]. University of Trier.

Verucchi, M., Olmedo, I. S., & Bertogna, M. (2023). A survey on real-time DAG scheduling, revisiting the Global-Partitioned Infinity War. *Real-Time Systems*, *59*(3), 479–530. https://doi.org/10.1007/s11241-023-09403-3

Yang, H., Zhao, S., Shi, X., Zhang, S., & Guo, Y. (2023). DAG Hierarchical Schedulability Analysis for Avionics Hypervisor in Multicore Processors. *Applied Sciences (Switzerland)*, *13*(5). https://doi.org/10.3390/app13052779

Yasmin, J., Wang, J. A., Tian, Y., & Adams, B. (2025). An empirical study of developers' challenges in implementing Workflows as Code: A case study on Apache Airflow. *Journal of Systems and Software*, *219*. https://doi.org/10.1016/j.jss.2024.112248

* Rama Aria Megantara