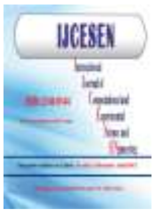
Copyright © IJCESEN

International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 12-No.1 (2026) pp. 470-478
http://www.ijcesen.com

ISSN: 2149-9144

Research Article



Can Small Teams Do MLOps Too? Starting Simple Without a Big Budget

Swati Kumari*

NucleusTeq, USA
* **Corresponding Author Email:** swati.kumari.reach@gmail.com - **ORCID:** 0000-0002-0047-660X

Article Info:

DOI: 10.22399/ijcesen.4833
Received : 28 November 2025
Revised : 26 January 2026
Accepted : 28 January 2026

Keywords

MLOps Implementation,
Resource-Constrained Teams,
Incremental Automation,
Experiment Tracking,
Lightweight Infrastructure

Abstract:

MLOps is viewed as a complex process for the enterprise level, so MLOps can serve as a significant hindrance for small teams who want to apply machine learning operations. Yet, small teams can gain a tremendous advantage from MLOps by applying simplified, lean tooling, gradually moving toward more complex MLOps for their teams. In this article, a complete set is presented for small teams on how MLOps can be applied effectively for small teams without engaging cloud orchestration platforms. The article discusses ideas on how MLOps can be applied for small teams through basic version control for source and model files, tooling for simple experiments with file storage and databases, basic automation through shell scripts, basic MLOps tooling through system job schedulers, and basic MLOPs testing through standard testing results. MLOps can be made a successful process for small teams with the use of a progressive approach. According to the progressive approach, teams can move toward more complex MLOps concepts when their skills and resource availability increase. Therefore, in the progressive approach, automation investments can provide a remarkable difference for teams, meaning investment in MLOps can be avoided because automation can provide a negative effect for the team. Therefore, even the leanest teams can attain a solid basis for successful MLOps.

1. Introduction

It has always been a dilemma among small teams working in resource-constrained environments how to ensure that their machine learning systems remain reliable without the budgets large technology companies have to invest in their systems. The advent of MLOps has become an essential element of solving modern ML development, yet poses an apparent impediment of the early adoption of more basic populations through its relationship to more complex tooling and cloud-native infrastructure. The intricacy of the full MLOps systems tend to overload staff who are already operating on the margins of core development tasks. Studies into the failures of big data and machine learning projects have reported systemic trends of organizations failing to move experimental models into working systems with the most common factors that have led to these failures being the lack of resources and insufficient operational practices [1]. Such problems are especially acute in the case of smaller organizations where employees of the team are required to combine several roles at once, and usually do not have the luxury of a dedicated staff focused specifically on infrastructure and deployment issues.

The apprehension is based on a primary misconception, i.e., that MLOps involves significant initial expenditure in platforms, human resources, and infrastructure. Research on the challenges of machine learning application has indicated that the barriers to the use of machine learning often occur at the level of complexity of deployment, monitoring needs, and the perceived complexity of complex orchestration systems that must be in place before the teams can put models into practice [2]. This perception produces a paralysis that the teams feel that they need to install enterprise-grade solutions initially or they will end up having systems that will not scale accordingly. The difference between the perception and the reality makes small teams delay the realization of the fundamental operation practices, which results in technical debt, reproducibility problems, and maintenance problems progressively increase. The accrued technical debt takes different forms: notebook-developed models that are not reproducible by other developers, computational resource-consuming experiments as a result of unintended duplication of prior work, manualized deployment processes that are subject to configuration drift and human error, and missing monitoring that does not allow detecting that model performance degrades in production settings.

As this article will demonstrate, it is possible to apply the MLOps principles incrementally and using the readily available resources and simple workflows in the context of the current capacity and requirements of the resource-bound teams. Beginning with these basic practices like version control of code and model setups, systematic test tracking with simple tools, rudimentary automated testing to identify regressions, and simple automation of deployment with simple scripts instead of engineering orchestration platforms, teams will realize immediate operational payback without a huge investment in capital or human resources. These baseline practices help in tackling the most significant pain points captured in the literature on machine learning deployment: reproducibility through a sound versioning mechanism, wasteful computation through proper experiment documentation, repetitive work with end users through the automation of routine work, and monitoring baselines with problem identification when an issue has been detected before it can impact end users [2]. Gradually increasing its capabilities and adding them as particular needs arise, instead of implementing extensive solutions speculatively, even lean teams can build strong ML operations to enhance daily productivity and position technical systems and organizational processes to grow in the future as projects become mature and resource availability deepens.

2. Understanding MLOps at Its Core

MLOps is basically an implementation of the software engineering field to machine learning processes. In its simplest form, it deals with three essential requirements: the ability to reproduce experiments and results, the ability to deploy a model reliably, and the ability to sustain an ML system over time. These are requirements independent of the size of the team and organizational resources. The literature review of the challenges and strategies of MLOps indicates that organizations of various sizes face the same underlying problems, i.e. versioning of models, repeatability of experiments, and consistency in terms of operations [3]. The reproducibility problem itself has become a universal issue in all organizations where training models constructed at previous project stages cannot be reliably recreated because the version of training data, hyperparameter settings, dependency descriptions or environmental settings are not properly documented. The reliability issue arises when models which perform well in controlled development environments fail to work in production environments owing to changes in data distributions, infrastructure setups, integration issues, or due to unexpected edge cases in actual data streams. The maintenance consideration grows in importance as the models grow older in the production process, and need to be updated systematically to accommodate the performance drift that occurs with changes in data patterns, inject new training data representing current conditions, meet the needs of changing business demands, or act upon feedback of prediction errors or bias.

MLOps is modular, which is obscured by the idea that the concept of MLOps necessitates enterprise-level solutions. Basic MLOps principles like version control, experiment tracking, test automation, and deployment automation can be applied separately and gradually. Small teams can use the fact that MLOps is not a binary condition but a continuum of practices that can be embraced given the existing needs and capabilities. There are quite a few works examining the machine learning lifecycle that demonstrate that a successful implementation of MLOps involves a number of steps: data collection and preparation, model development, deployment, and monitoring, each of which can be enhanced by incremental operations [4]. This planned development gives teams the freedom to focus on those practices that will eliminate the most urgent sources of pain instead of seeking an overall change simultaneously. Modularity implies that version control can be set up on the basis of standard distributed version control systems requiring no special machine learning platforms; that experiment tracking can be initiated using the basic shell scripting or scheduled tasks before progressing to more advanced container orchestration systems or serverless computing architectures. Beginning with the simplest viable practices achieves value in the present and establishes complex-accommodating patterns in the future. Code and data configuration Version controlling prevents (irreversible) loss of working solutions in which the teams invested substantial time and computation resources in developing. Systematic experiment tracking eliminates the wasted computation of repeating a prior experiment, which is becoming more and more important as model architectures grow more complicated, training durations grow beyond hours to days, and cloud computing is getting more expensive with each experiment. Automation minimizes the number of people errors that cause production accidents, failure of services to the end users or wrong forecasts undermining stakeholder confidence in machine learning systems [3]. They both have independent gains as they lead to overall operational maturity. The ensemble effect of these practices at the ground level is that small investments in operational discipline will translate into large returns in terms of productivity, better interaction among team members of different skills levels, faster iteration cycles that will enable more experiments and innovations, and increased reliability of a system, which creates organizational trust in the implementation of the machine learning solutions in critical production settings [4]. Such compounding advantages provide reasons to invest more in moving to the right of the MLOps maturity spectrum by adding teams and capabilities and expanding resources.

3. Building Blocks for Budget-Conscious MLOps

The very core of any available implementation of MLOps is version control systems. In addition to a versioning of code changes, proper version control covers model architectures, training settings, data processing programs and environment requirements. Branching strategies allow teams to become experimental but have a stable production code. Tagging releases gives an explicit view of the deployed models that allow rollbacks in the case of any problems. Studies on the support of MLOps tools take note of the fact that version control is the fundamental component on which all the other operational capabilities are based and which offers necessary traceability of reproducible machine learning systems [5]. It is not just important in traditional software version control terms that the machine learning system must rely on numerous interrelated components, which must be versioned in a consistent manner: training code implementing algorithms, hyperparameter settings with enormous effects on convergence and performance properties of the model, feature engineering pipelines converting raw input data to model-ready formats, model architecture description detailing layer configurations and connectivity patterns, dependency specifications giving a consistent execution environment both in development and production, and extensive metadata indicating the conditions under which the model was trained, such as hardware settings and random seed values. Teams which adopted a comprehensive version control of all these components have demonstrated that they were much better able to recreate experimental results many months after first being written, debug production problems by comparing deployments with previous versions in a systematic manner, and induct new team members who can perceive system evolution and design decisions by accessing commit history and documentation.

Data versioning also seems to be an issue, given the typical file sizes, yet there are lightweight methods. Storing snapshots of data as compressed archives, maintaining metadata manifests that characterize versions of datasets, or content-addressable storage patterns all can provide traceability with no special hosting facilities. It is all about building systematic naming conventions and documentation practices that render dataset lineage visible. End-to-end machine learning pipeline analysis studies in the cloud have found data versioning to be an important, but frequently ignored, feature of MLOps implementation, which explicitly influences model reproducibility and debugging ability 6. Training datasets are often measured in gigabytes or terabytes, and thus naive use of standard version control systems is not possible because they bloat over time and cause poor performance when performing repository operations, as well as have implications on storage cost. Lightweight approaches avoid these limitations by making strategic trade-offs: the size of archives is reduced by compressing snapshots, but full snapshots are not, at significant milestones of the project; metadata manifests can reveal significant properties of datasets, such as counts of rows and columns, distributions of features and transformations applied, but not duplicate the underlying data files; deduplication-unchanged data segments are only stored once, across multiple versions; and cryptographic hash-based verification can ensure data safety, even in the absence of special hosting infrastructure and/or rich validation processes.

Tracking Experiments Without Complicatedness.

Formal monitoring of experiments is used to substitute informal development of models with a model development procedure. Open-source tracking tools allow teams to record parameters and metrics and artifacts on a regular basis in experiments. This creates a searchable history which eliminates duplication of work and accelerates iteration by ensuring successful settings can be retrieved quickly. Investigations on the ecosystems of MLOps tools observe that experiment tracking addresses one of the oldest dilemmas of machine learning development, namely the rapid expansion of experimental variations that soon cannot be controlled unless orderly arranged and reported [5]. Teams that execute hyperparameter searches of hyperdimensional parameter spaces, that are comparing alternative neural network models with varying depth and width configurations, or that are comparing alternative feature engineering methods generate dozens or hundreds of experimental executions with each having different configurations that gave different performance properties across many evaluation metrics. Without organized recording systems, useful information on what strategies are effective in which circumstances will be stored in solitary Jupyter notebooks, fragmented log files, or undocumented mental models of a single contributor; a lot of redundant work will be done when different members of the team explore identical settings that have already been tried or be unable to find the same promising results that were discovered at an earlier stage of development.

Good tracking does not require advanced infrastructure. File-based backenders store experiment data on a local or shared network storage and are independent. Simple databases give queryable experiment history, obtained by structured logging to them. The use of a spreadsheet based tracking is of value when it is done in a disciplined way, though task specific tools saves more overhead and enhance uniformity. Various works have examined the machine learning pipeline implementations and have determined that experiment tracking can bring valuable operational advantages to teams with low infrastructure costs [6]. Small teams are adequately served by file based backends, which store structured metadata of the experiments on local filesystems or network-attached storage. These enable simple queries of experiments and performance comparisons without the need to have specific server infrastructure, expertise in database administration, and constant subscription to cloud services that add overhead costs to operation.

4. Automation Strategies for Small Teams

4.1 Starting with Simple Pipelines

Simple Pipelines is a simple initial pipeline system that functions according to the principle that water's specific gravity is greater than that of oil.<|human|>Simple Pipelines: beginning with Simple Pipelines. Simple Pipelines is a simple type of initial pipeline system which operates based on the principle that water has a higher specific gravity than oil.

Automation should not start with complicated orchestration systems. Task schedulers at the system level offer predictable automation to performing routine workflows. Training pipelines may also be scheduled to run or activated by signals of data availability. Evaluation scripts can be run automatically on held-out test sets to provide performance reports by which teams can detect degradation. Studies discussing continuous integration and continuous delivery as a means of automated deployment of machine learning models show that simple automation strategies provide significant value without implying complex infrastructure or tooling ecosystems development [7]. Task schedulers on the system level that are provided by all operating systems allow a team to create consistent periodic execution of training processes, data preprocessor, and model evaluation processes that are not deployed on specific orchestration platforms, do not require the implementation of complex workflow definition languages, and do not need to maintain other infrastructure elements. By training pipelines to be run at off-peak times of the computation, the maximum use is made of the available hardware resources, and in interactive development activities, when a model needs to be triggered to run to absorb new information is better done by use of trigger based execution mechanism that reacts to signals of data availability by the upstream source systems. Systemic execution of automated evaluation scripts on held-out test datasets yields stable performance reports that can be used to set behavioral expectations of model predictions, so that teams can identify performance decay caused by drift in data distribution, accidental software regressions, or environmental configurations before the problems can spread to production prediction services and affect business processes or user experiences.

Chaining Shell scripts, reproducible pipelines can be generated by chaining together preprocessing, training, and evaluation steps, which anyone on the team can use. The execution logic is separated by configuration files with which these scripts are parameterized and experimental parameters, thus allowing easy modification. This is a non-architectural scaling of local development machines to common compute resources. Research on the difficulties surrounding the creation and implementation of artificial intelligence models in industrial environments has emphasized script-based pipelines as a welcoming starting point to teams embarking on automation as it offers instant reproducibility advantages without the high learning curve of the complicated workflow engines or dedicated orchestration-focused tools [8]. Sequences of data preprocessing operations, model training processes, validation routines, and evaluation programs are shell scripts describing key institutional knowledge of the order of correct execution, input file locations and formats, output artifacts and specifications, dependency management process, and error handling routines which would otherwise be stored only in the unwritten memory of individual developers or on wiki pages and email archives. By parameterizing them with external configuration files in standard formats, such as JSON or YAML, the teams can change hyperparameters that affect the model behavior, data source locations or selection criteria, evaluation metrics or reporting formats, or number of computational resources but without changing the core scripts logic and thus reduce significantly the risk of syntactic errors or logical bugs to the point of making changes during normal experimentation cycles.

4.2 Progressive Enhancement

The level of automation is an organic development that follows the demands of a group. Planned scripts are replaced by event-based triggers as workflows become more fluid and manual steps of the deployment process are replaced by automatic ones as trust is established. Monitoring capabilities vary depending on the complexity of a system between the simplistic logging to structured metrics gathering. Each improvement is very specific, instead of affecting capabilities in a speculative fashion. Studies of continuous integration and delivery of machine learning have shown that gradual optimization of automation, ensuring that all automation efforts are well-calibrated with real operational requirements and the capacity of the team, leads to increasingly sustainable implementations, as opposed to aiming to achieve full end-to-end automation too soon. Teams are generally embarking on an automation journey with simple fixed-cadence scheduled execution of training workflows and then gradually move on to event-driven architectures in which data availability announcements by upstream systems, observable performance thresholds by monitoring, or manual announcements by data scientists may trigger pipeline execution as the business needs of more responsive adaptive systems become evident. On the same note, manual processes of deployment that involve human verification, approval gates, and rollout processes are replaced with automated deployment pipelines that also consist of full testing suites and automated validation checks as the different teams continue to get more operational experience with model behavior in production scenarios, develop confidence with their quality assurance processes and develop robust rollback mechanisms that effectively address the risks inherent in fully automated releases.The improvements are focused on particular pain points, which in practice are not only stalling the workflows but also introducing the capabilities' implementation according to the speculation of the ideal work that should be. It makes sure that automation investments can provide operational returns instead of increasing system complexity and not productivity returns. The study examining issues in the deployment of industrial AI proved that it is clear that the teams achieving sustainable automation maturity are strategically concerned with the improvement of the real bottlenecks that restrict their growth pace or functional stability, as opposed to the implementation of features as predetermined by the generic MLOps framework or futuristic reference architecture [8].

5. Testing and Validation Practices

Testing discipline is crucial from the outset to avoid quality problems from accumulating. Unit tests assess the correctness of the logic on which the data processing is based. Integration tests check the functionality of various components within the processing pipeline. Tests for validating models check their predictions on standard cases and their efficiency on different data segments. Such activities can be accomplished with a bare minimum of infrastructure, relying on generic testing tools, but they have to be performed consistently. Studies on the perspectives on the results related to the incorporation of artificial intelligence and machine learning techniques on the generation and accumulation of debt cite the necessity for the application of a multifaceted approach toward complete testing toward the maintenance of the quality features within the studied models, irrespective of the progressive evolution and change in distributions within the analyzed data sources through time [9]. Individual unit tests related to the processing functions assess the functionality when the feature transformation logic is applied, the functionality when the cleaning logic based on the input and output is applied, the functionality when the preprocessing logic is applied based on the transformation chains, and the functionality when the preprocessing logic is applied based on the transformation chains. Integration tests applied on a more abstract level confirm the functionality when the components interact within the complete workflow, assess the functionality when the output format from the preprocessing stage corresponds with the input expectations for the training modules, the functionality when the model's deserialization and serialization logic maintains the prediction functionality on various runtime environments, and assess the functionality when the evaluation harnesses appropriately use the model’s outputs for the metric computation. The tests for validating models work on the highest level and assess the functionality upon the prediction on standard cases based on the established outputs within the historical data sources, the functionality upon the efficiency based on the different segments within the defined data sources based on demographic information, and the functionality upon the various input perturbations based on the real-world variations.

These methodologies require low-barrier tooling, as the testing infrastructure already supported by all programming languages will suffice, albeit with rigorous adherence throughout the entire development cycle for maximum liable-end protective benefits. Technical debt research and analysis studies among AI-based systems clearly indicate the testing discipline paradigm as a paradigm where pooling cumulative benefits over time requires initial heavy spending on comprehensive testing suites to avoid protracted debugging processes and production-level events, with later analyses otherwise requiring far more resources and expense to debug and rectify [10]. The tool infrastructure remains low-barrier because the current unit testing infrastructure already exists and was devised for traditional softwares has a direct natural extension for AI-based machine learning systems requiring only the addition of standards for representing test cases, wherein acceptable margins for numerical testing comparisons subject to floating point arithmetic precision and stochastic training processes must also require, and representing fixtures for test data with diverse minutiae without making them unmanageable for large-scale systems, whereas the key to success remains rigorous adherence and not the complexity and infrastructure overheads requiring standardization where each code change should include tests, wherein test failures are automatically and automatically blocked for integration into the development trunk, and wherein developing and maintaining test infrastructure health should rank on par with new feature enhancements or boosted accuracy for AI-based systems as well.

Lightweight continuous integration through a code repository hook or timed validation runs identifies bugs prior to deploying them in production environments. Automated test scripts executing on each code change offer fast in-process feedback loops during software development activities. Staging environments that are simply separate config files directing references to non-production data sources provide a safe setting prior to software deployments or updates in production environments using this solution that costs nothing but significantly impacts reduced frequency or occurrence in production environments due to reduced software issues or bugs. A study from research in AI/Machine Learning infrastructure development explicitly addresses that automating software tests integrated into a software development process boosts immediate feedback loops in debugging activities that fast-track bug fix resolutions in systems while avoiding regressions from a software update or modification in a system that increases developers' confidence in a system whose functionality operates as expected [9]. Repository hooks that automatically trigger test execution scripts upon each code change ensure that software bugs are quickly identifiable in minutes from their code introduction during a setting that ensures minimal remediation work due to fresh implementation contexts in system developers' minds compared to environments requiring delayed validation that involve extensive code changes from a bug introduction point prior to system visibility in production environments for increased remediation difficulties in search and isolation activities due to their buried system contexts among multiple code changes in rapid software changes in a system environment.

Such practices have minimal implementation costs and mostly depend on the discipline within an organization as well as the computational resources necessary to run tests, while they greatly minimize the occurrence of issues within the production environment as most issues are identified before the systems go to the user environment. Findings from research that focus on methodologies of technical debt management have identified that organizations that adopt total testing and validation methodologies experience fewer failures within the production environment and reduced mean time to detect and repair when failures occur [10].

***Table 1**: MLOps Core Components and Implementation Characteristics for Small Teams [3, 4]*

MLOps Practice	**Primary Purpose**	**Initial Implementation Approach**	**Progressive Enhancement**	**Team Impact**

Version Control	Reproducibility of experiments and models	Standard distributed version control systems for code and configurations	Add data versioning through metadata manifests and content-addressable storage	Prevents loss of working solutions; enables collaboration
Experiment Tracking	Eliminate duplicate work and enable configuration retrieval	Structured logging to simple databases or file-based backends	Adopt dedicated tracking servers with visualization capabilities	Reduces wasted computational effort; accelerates iteration
Automated Testing	Reliability and quality assurance	Unit tests for data processing; integration tests for pipeline components	Expand to model validation tests and continuous integration hooks	Reduces production incidents and service disruptions
Deployment Automation	Maintainability and error reduction	Basic shell scripting or scheduled tasks	Progress to container orchestration or serverless architectures	Minimizes human errors; frees time for model improvement

Table 2: *Version Control Components for MLOps [5, 6]*

Component	File Size	Implementation Method	Key Benefit
Source Code	Megabytes	Standard Git repositories	Code reproducibility
Hyperparameters	Kilobytes	Configuration files (JSON/YAML)	Parameter tracking
Training Data	Gigabytes to Terabytes	Compressed archives + metadata	Data lineage
Model Architecture	Kilobytes	JSON/YAML specifications	Architecture versioning
Dependencies	Megabytes	Requirements files	Environment consistency

Table 3: *Automation Maturity Evolution Path [7, 8]*

Maturity Stage	Trigger Mechanism	Deployment Process	Monitoring Level	Team Confidence	Implementation Focus
Initial	Manual execution	Fully manual with verification	Basic logging	Low	Establishing scripts
Basic	Fixed schedules	Manual with checklists	Structured logs	Building	Reliability improvement
Intermediate	Event-driven triggers	Semi-automated with approvals	Metrics collection	Moderate	Reducing manual steps
Advanced	Intelligent triggers	Fully automated with rollback	Comprehensive dashboards	High	Performance optimization

Table 4: *Testing Practice Implementation and Impact Analysis [9, 10]*

Testing Approach	Setup Effort	Organizational Requirement	Feedback Speed	Impact on Production Incidents	Long-term ROI
Manual Testing	Minimal	Individual discipline	Hours to days	Moderate incident rate	Low
Basic Automated Tests	Low	Team conventions	Minutes to hours	Reduced incidents	Moderate
Repository Hook Testing	Moderate	CI/CD setup	Minutes	Significantly reduced incidents	High
Comprehensive Test Suites	High	Strong testing culture	Real-time	Minimal incidents	Very High

6. Conclusions

Small teams with low budgets can practice effective MLOps by accepting the fact that operational maturity must instead be measured on a spectrum, rather than seeing the process of implementation as an all-or-nothing solution for the enterprises. The trick here is that a set of skills needs to be incrementally introduced to the system, focusing on the basic practices of version control, experiment tracking, basic automation, and test discipline. Indeed, all of these basic practices are important in themselves, but together, they provide a starting foundation for the necessary set of skills for operation infrastructures in machine learning. The tools used within the system include basic version control tools, basic tracking tools, basic scripts for the logistics of the system and the automation of the process, and basic test tools, all of which can be used within the system to address the critical needs regarding the reproducibility, the interaction within the system, and the reliability of the deployment process with the system in place, all of which can be done with less budget within the system. Moreover, the ideology of progressive enhancement allows the system to develop the level of automation maturity increments, in line with the growth of the system's needs and possibilities in the process of implementation of practice within the system's framework. In other words, each of the progresses developed within the system aims to resolve the existing bottlenecks in the system, rather than focusing on the assumed needs for the system according to the ideal framework of the system's practice, processes, and operations, within the specified process in the system's framework. Indeed, the enterprises that adopted the practical approach to implementation of the practice in the system have developed numerous benefits within the system's framework and process, including the reduced technological debt of the system, the reduced incidents of production within the system, an increased level of efficiency of the process for the system's validation, an increased level of cooperative processes within the system's framework for the validation process, the reduced time of the system's iteration, and the increased confidence within the system's validation for the machine-learning-oriented processes in the system's framework, all of which clearly indicate that the implementation of the practice of the system's operation does not require significant budget within the system's framework. Indeed, the necessary budget within the system's process for the implementation of the practice includes the systematic discipline of the organization within the system's framework, the practice of all the system's operational processes in the system's framework, and the systematic selection of the tools within.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] Pouya Ataei et al., "Why Big Data Projects Fail: A Systematic Literature Review," International Journal of Information Management Data Insights, January 2025. [Online]. Available: https://www.researchgate.net/publication/388038922_Why_Big_Data_Projects_Fail_A_Systematic_Literature_Review

[2] Alexandra Clara, "A Survey of Applications, Challenges, and Future Directions in Machine Learning," ResearchGate, February 2025. [Online]. Available: https://www.researchgate.net/publication/389659114_A_Survey_of_Applications_Challenges_and_Future_Directions_in_Machine_Learning

[3] Amandeep Singla, "Machine Learning Operations (MLOps): Challenges and Strategies," International Journal of Advanced Computer Science and Applications, vol. 15, no. 1, August 2023. [Online]. Available: https://www.researchgate.net/publication/377547044_Machine_Learning_Operations_MLOps_Challenges_and_Strategies

[4] Zhengxin Fang et al., "MLOps: Spanning Whole Machine Learning Life Cycle, A Survey," arXiv preprint, April 2023. [Online]. Available: https://www.researchgate.net/publication/370070459_MLOps_Spanning_Whole_Machine_Learning_Life_Cycle_A_Survey

[5] Nipuni Hewage & Dulani Meedeniya, "Machine Learning Operations: A Survey on MLOps Tool Support," arXiv preprint arXiv:2202.10169, February 2022. [Online]. Available: https://www.researchgate.net/publication/358766274_Machine_Learning_Operations_A_Survey_on_MLOps_Tool_Support

[6] Lee Michael et al., "End-to-End ML Pipelines in Cloud Environments for AI-First Product Engineering," ResearchGate, June 2023. [Online]. Available: https://www.researchgate.net/publication/39570509

0_End-to-End_ML_Pipelines_in_Cloud_Environments_for_AI-First_Product_Engineering

[7] Satvik Garg, "On Continuous Integration/Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps," ResearchGate, December 2021. [Online]. Available: https://www.researchgate.net/publication/359000282_On_Continuous_Integration_Continuous_Delivery_for_Automated_Deployment_of_Machine_Learning_Models_using_MLOps

[8] Sudhi Sinha & Young M. Lee, "Challenges with developing and deploying AI models and applications in industrial systems," Software and Systems Modeling, August 2024. [Online]. Available: https://www.researchgate.net/publication/383198725_Challenges_with_developing_and_deploying_AI_models_and_applications_in_industrial_systems

[9] Dimitri Kalles, Dionysios Sklavenitis, "A Scoping Review and Assessment Framework for Technical Debt in the Development and Operation of AI/ML Competition Platforms," arXiv preprint arXiv:2410.20199, June 2025. [Online]. Available: https://www.researchgate.net/publication/393087944

[10] Gilberto Recupito et al., "Technical debt in AI-enabled systems: On the prevalence, severity, impact and management strategies for code and architecture," Journal of Systems and Software, July 2024. [Online]. Available: https://www.researchgate.net/publication/382011632