DOI: 10.15514/ISPRAS-2026-38(3)-8

MLManagement: automating ML pipelines with reusable abstractions and access control

M.A. Ryndin, ORCID: 0000-0002-7504-3975 <mxrynd@ispras.ru>
A.M. Boiko, ORCID: 0009-0005-2671-5551 <boeing@ispras.ru>
D.O. Kushchuk, ORCID: 0000-0002-8778-8608 <dkuschuk@ispras.ru>
D.M. Kiranov, ORCID: 0000-0002-3507-3803 <kiranov.dm@ispras.ru>

Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. The current Machine Learning (ML) landscape requires engineers to train and deliver models robustly, timely and at scale to stay competitive. Time to market could be reduced if specialists had a tool that would automate and address ML engineering challenges specific to the ML workflow such as experiment tracking, model versioning, etc. Although existing open-source MLOps tools provide functionality that partially covers these needs, most of them have limited access control and experiment code reuse capabilities. To this end we introduce the MLManagement platform. Our main contributions are as follows: firstly, we introduce reusable building block abstractions that enable users to quickly create automated ML experiments in a "plug-and-play" fashion. Secondly, we encourage collaboration in big teams or multi-team organizations by implementing Attribute-Based Access Control (ABAC) for all the platform's entities such as models, artifacts and data, as well as hardware resources allocation.

Keywords: MLOPS; machine learning; access control.

For citation: Ryndin M.A., Boiko A.M., Kushchuk D.O., Kiranov D.M. MLManagement: automating ML pipelines with reusable abstractions and access control. Trudy ISP RAN/Proc. ISP RAS, vol. 38, issue 3, part 1, 2026, pp. 143-152. DOI: 10.15514/ISPRAS-2026-38(3)-8.

MLManagement: автоматизация ML конвейеров с переиспользуемыми абстракциями и контролем доступа

М.А. Рындин, ORCID: 0000-0002-7504-3975 <mxrynd@ispras.ru>
А.М. Бойко, ORCID: 0009-0005-2671-5551 <boeing@ispras.ru>
Д.О. Кущук, ORCID: 0000-0002-8778-8608 <dkuschuk@ispras.ru>
Д.М. Киранов, ORCID: 0000-0002-3507-3803 <kiranov.dm@ispras.ru>

Институт системного программирования им. В.П. Иванникова РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.

Аннотация. Текущее состояние индустрии машинного обучения (Machine Learning, ML) требует, чтобы процессы обучения и развертывая моделей были надежны, своевременны и масштабируемы для сохранения конкурентоспособности. Время вывода решений на рынок может быть существенно сокращено, если специалисты будут обладать инструментом, который автоматизирует специфические задачи, возникающие в процессе получения ML моделей, такие как отслеживание экспериментов, версионирование моделей и другие. Несмотря на то, что существующие инструменты MLOps с открытым исходным кодом частично покрывают эти потребности, большинство из них обладают ограниченными возможностями контроля доступа и переиспользования кода экспериментов. Чтобы решить эту проблему, мы представляем платформу MLManagement. Наш основной вклад заключаются в следующем: во-первых, мы вводим переиспользуемые абстракции-строительные блоки, позволяющие пользователям быстро создавать автоматизированные ML-эксперименты по принципу "plug-and-play". Во-вторых, мы способствуем улучшению коллаборации в крупных командах и организациях с несколькими командами, реализуя модель контроля доступа на основе атрибутов (Attribute-Based Access Control, ABAC) для всех сущностей платформы – моделей, артефактов, данных, а также для распределения аппаратных ресурсов.

Ключевые слова: МЛОПС; машинное обучение; контроль доступа.

Для цитирования: Рындин М.А., Бойко А.М., Кущук Д.О., Киранов Д.М. MLManagement: автоматизация ML конвейеров с переиспользуемыми абстракциями и контролем доступа. Труды ИСП РАН, том 38, вып. 3, часть 1, 2026 г., стр. 143–152 (на английском языке). DOI: 10.15514/ISPRAS–2026–38(3)–8.

1. Introduction

Advancements in the field of ML have driven the adoption of ML algorithms in large-scale commercial IT services and applications. ML is widely adopted in domains of personalized advertising, healthcare, self-driving transport, etc. The development of ML-driven software and its integration into production environments, however, raises unique challenges that are not typical for conventional software. To enable ML engineers to produce ML models in a reliable and automated fashion, the methodology called MLOps was developed. It is often conceptualized as an extension of the DevOps methodology to include ML assets as its first-class citizens and tackle ML-specific challenges [1].

Specific MLOps principles arise from ML's experimental and data-driven nature [2]. Large volumes of training data need to be effectively stored and managed. Training of an ML model is largely a trial-and-error affair, requiring many experiment runs. Each experiment has to be reproducible and traceable (to help guide the research, as well as for compliance and auditing reasons). To ensure it, the experiment metadata, the resulting model, its training data, its artifacts and all the additional code has to be versioned and stored with proper access control to foster inter and intra team collaboration. Experiment results and metrics should be visualized and compared in useful dashboards. ML experiments need to be properly planned, as they are computationally intensive, utilize specific hardware such as GPUs and TPUs and take a long time to complete. When using a computational cluster in a multi-user setting, its hardware resources should be automatically distributed to users in accordance to their requirements and privileges in order to utilize the resources

effectively. Experiments should be run in isolated environments for safety and scalability reasons. Experiment jobs should be observable in real-time. This illustrates how complex the mature MLOps pipeline has to be.

Failure to incorporate any of the required steps results in growing technical debt and significantly increases time to market [3]. MLOps pipelines are tedious to implement and require ML engineers to attain skills from other software development fields such as DevOps and network engineering, which is distracting from their primary role [4]. Thus, developers of MLOps tools seek to implement and automate MLOps pipelines to lighten the burden on ML engineers. To this end, our tool allows users to create automated ML experiments in just a few clicks via reusable code abstractions while fostering collaboration via access control.

2. Related work

There exist many popular open-source MLOps tools. Since a comprehensive survey on the MLOps tools is out of scope of this paper, we will instead give an overview of a number of common open-source MLOps tools that we have experience with, namely MLFlow, DVC, Kubeflow and ClearML. We will focus on their collaboration features and ML experiment automation capabilities, i.e., running arbitrary code in isolated environments as planned asynchronous jobs.

MLFlow [5] is mainly an experiment tracker which allows for model versioning, artifacts logging and artifacts comparison and doesn't cover the whole MLOps pipeline, e.g., no experiment automation capabilities are supported. Data management and online system monitoring are lacking. It also uses unsafe Pickle serialization [6] for model storage and is disk inefficient. Collaboration features are limited to authentication and individual user permission granting for experiments and models. User groups are not supported.

DVC was originally designed for data versioning, but has since added a number of MLOps functionalities such as experiment tracking. It requires integration with code repositories' (such as GitLab) cloud job runners to automate ML experiments and therefore automation and collaboration capabilities are identical to those of the DevOps platform it is integrated with. DVC is still mainly used for data versioning and compression due to difficulties with using multiple models in a single repository, the need for configuring runners via YAML and the lack of priority-based runner resource allocation when used in multi-team or multi-repository settings (thus hindering collaboration).

KubeFlow is a powerful tool for automating ML pipelines via the Kubernetes orchestrator and Docker containers. It is, however, inconvenient for ML engineers, as it requires DevOps skills to operate. It also lacks functionalities for data management and experiment result analysis. KubeFlow allows users to build arbitrary directed acyclic graph pipelines of Python code with manual or periodic triggers. Authentication and authorization are present, but synonymous to Kubernetes roles. User isolation is namespace-based, which presents security risks.

ClearML is a comprehensive MLOps pipeline tool. It covers a broad range of functionalities such as experiment code automation, data management and system monitoring. Provided pipeline building tools are powerful but use Pickle to pass entities between steps, presenting security issues and limitations [6]. Planning and authentication features are very limited in open-source versions, thus hindering collaboration capabilities.

Although DVC, KubeFlow and ClearML do support experiment code automation, they do not offer any explicit experiment code reusability features. If users want to reuse any of their code, e.g., use a certain piece of code implementing some common ML functionality across different experiments, they have to design such features themselves.

We now present qualitative comparison of our tool with aforementioned popular open-source tools. The comparison presented in Table 1 is inspired by [1], but for the sake of brevity we chose shallower feature taxonomy to focus on the tools' integration into the work of ML engineers in mid-scale organization.

Table 1. Qualitative comparison of MLOps tools.

Tool name	Introduction/ Use Case	Data management	Model management	Experiment management	Monitoring	System design
MLFlow	Experiment tracking and model registry	o	++	+ EM-4 EM-3 EM-5	o	++
DVC	Data versioning tool with additional MLOps features	+ DM-1 DM-3	+	+ EM-4	o	+
KubeFlow	Integrating ML pipelines with Kubernetes orchestrator	o	++	+ EM-3 EM-1 EM-5	++	++
ClearML	Complete model lifecycle	++	++	+ EM-1 EM-3 EM-4 EM-5	++	+
MLManagement (ours)	Complete model lifecycle with reusable abstractions and access control	+ DM-1 DM-2	++	++	++	++

We benchmark the tools against the following feature groups:

1) Data management workflow with the ability to integrate with different data sources and storage types (DM-1), preprocess data during experiment (DM-2) and support data versioning (DM-3);
2) Model management workflow with model versioning, including model artifacts and source code;
3) Experiment management with experiment execution planning and isolated container orchestration (EM-1), experiment code reusability (EM-2), python framework agnosticism (EM-3), tools for cross-experiment result analysis and comparison (EM-4) and experiment reproducibility (EM-5);
4) Monitoring, including system monitoring with online tracking of logs, statuses and hardware resource utilization;
5) System design, including the ability to integrate with other systems to extend existing workflows, maturity of tools and libraries used and possibility of simultaneous usage by more than a few users.

Each tool is given a ranking for feature group support, where "o" denotes that feature group is missing, "+" and "++" denote partial and full support respectively.

Each described tool offers limited collaboration features, and experiment code reusability is left for the users to implement, even if experiment automation is supported. We aim to lighten this burden by implementing experiment code reusability as part of our MLManagement platform, as well as providing mature collaboration features.

3. Problem statement

We find that existing open-source MLOps tools have the following main limitations:

1) **Limited collaboration ability**, hindering models and artifacts sharing [1] and knowledge transfer, as well as complicating the sharing of computational resources between team members;
2) **Limited experiment code reusability** within automated ML experiments, leading to cumbersome experiment creation process, code duplication and difficulties with updating existing experiments.

To address limited collaboration ability, we take the following actions as part of our MLManagement platform:

1) Implementation of **ABAC** [7] to all the platform's entities, as access control to ML models and training data is essential for assuring safety [8] and preventing data leaks [3, 8-9], and **ABAC** is one of the most flexible access control models, which enables entity access

decisions to be made via comparing the attributes of users and target entities;

2) Implementation of **hardware resource allocation** to prevent unavailability of computational resources necessary for ML experiments (such as GPUs and TPUs) in a multi-user setting.

To address limited experiment code reusability, we take the following actions as part of our MLManagement platform:

1) Implementation of **reusable abstractions**, enabling quick creation of automated experiments using replaceable "plug-and-play" building blocks;
2) Implementation of a common **reusable abstraction registry** with **ABAC** to enable users to reuse and extend abstractions, as well as foster collaboration;
3) Implementation of an **experiment constructor UI** with automated compatible abstraction filtration to enable experiment creation in just a few clicks.

We evaluate our platform's collaboration features in a multi-user multi-experiment setting, measuring its impact on disk utilization and platform latency in section 5.

4. MLManagement platform overview

In Fig. 1 an MLOps pipeline implemented using our MLManagement platform (on the left) is compared to an MLOps pipeline implemented using other mentioned tools (on the right). Our platform enables collaboration via ABAC to all entities (portrayed via blue arrows). Users construct experiments via selecting appropriate reusable abstractions from the registry, which are then passed into an isolated environment together with data from the integrated storage for automated planned execution on hardware resources distributed to users by Slurm. This is in contrast to other tools built around non-reusable experiment code, where each experiment needs to be fully described in code each time. Moreover, other tools might require integration with different tools in order to fully support the illustrated pipeline, which is portrayed via greyed out elements on the right. Other tools also generally have limited access control in open-source versions (portrayed via gray arrows).

We will now discuss the architectural implementation of access control in our platform. Then, we will discuss the design and interfaces of reusable abstractions of the platform.

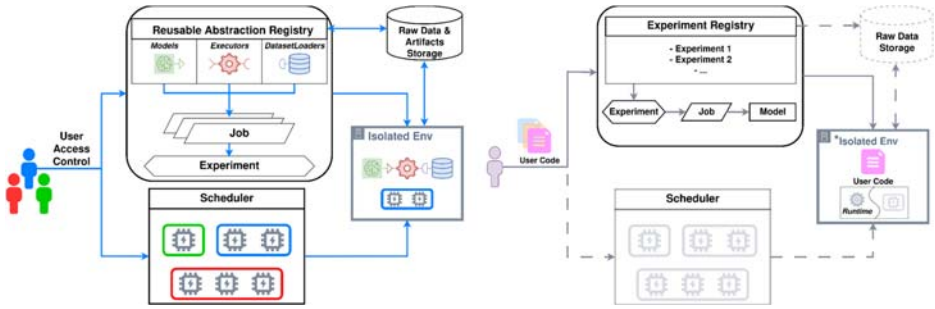

Fig. 1. MLManagement (on the left) enables collaborative MLOps pipelines with quick experiment creation via reusable building block abstractions. Other tools (on the right) have limited collaboration features and do not natively support experiment code reusability.

4.1 Access control and hardware resource allocation

Implementation of ABAC [7] to all the MLManagement platform's entities, including common reusable abstraction registry, as well as automatic hardware resources allocation, allows to address existing open-source MLOps tools' limited collaboration ability while ensuring safety of ML models and training data.

Differentiating the access to entity registries presents an issue, as registries are returned in the form of paginated lists. This means that inaccessible entities need to be filtered out on the database level before accessible entities are organized into pages, as filtering paged entities would result in incomplete pages and the need to make an unknown number of additional requests (and filtering) to fill them. Usually access policy trees are described by XACML standard that can be used to filter only individual resources.

To this end, we have developed a microservice that stores and evaluates specific attribute-based access policy trees. These policy trees are conceptually similar to XACML standard, with each node containing rules that compare various attributes and get translated into an arbitrary domain-specific language (DSL), such as SQL. A key distinction from standard rule trees lies in our ability to partially reduce policy trees to DSL, instead of reducing them to a single boolean value. For example, the policy tree "*@model.author = @user.id or @model.is_public = true*" gets reduced to an SQL expression "*author = John Doe or is_public is true*". This allows us to augment a paginated query with a DSL sub-expression that filters out entities that are inaccessible for a particular user prior to pagination, thus enabling effective ABAC to paginated registries, as well as single entities. The lifecycle of any request consists of user attributes retrieval and partial reduction of the target policy tree to an SQL expression, which is then added to the database request that is performed to retrieve the target entity.

As a result, access to all platform entities including registries becomes fully authorized, giving system administrators the ability to set up custom fine-grained access policies via policy trees editing, thus fostering collaboration and assuring safety.

Another facet of access differentiation is automatic hardware resources allocation. We achieve it via incorporating the Slurm workload manager [10], which supports privilege-based hardware resources allocation. User's Slurm group is stored as a user attribute that is added to the planning request. This enables system administrators to grant different privileges and quotas to different team members, allowing for effective hardware resource utilization in a multi-user setting and enabling engineers to collaboratively use a single computational cluster.

4.2 Reusable abstractions

In this subsection we will discuss the design and interfaces of the proposed reusable building block abstractions in our MLManagement platform: Executor, Model and Dataset Loader, as well as the process of constructing an experiment using these abstractions.

Reusable abstractions are meant to alleviate the limited experiment code reusability issue present in other MLOps tools, where users are required to write all the code which comprises each experiment, thus limiting the ability to reuse basic ML experiment elements such as models, data preprocessing pipelines and other common code across different experiments. We employ the bottom-up strategy, where each experiment is constructed from reusable building block abstractions, namely: Models, Dataset Loaders and Executors. This approach enables creating experiments in a few clicks without writing any code, just by using the Experiment constructor UI (given that the required abstractions are already uploaded onto the platform). As models and data preprocessing pipelines are presented as reusable "plug-and-play" abstractions, they can be easily swapped out in the same experiment setting, enabling ML engineers to quickly test different hypotheses. Vice versa, experiment settings can be divided into common types and made into reusable Executor abstractions, which allows swapping out experiment settings using the same models and datasets depending on the task. These common types of ML experiment settings include model training and finetuning, as well as model evaluation, pruning, distillation, quantization, data anonymization, model standardization before deploy, etc.

The ability to reuse experiment code abstractions in combination with access control allows for better collaboration, less laborious experiment creation and updating processes and less code duplication. Let's examine each abstraction in detail.

4.2.1 Executor

An Executor is an abstraction that defines the general experiment design and manages all the processes involving models and data via a set of typed interface methods. An Executor defines a sequence of arbitrary actions using an arbitrary set of Model and Dataset Loader abstractions, as long as these abstractions implement the Executor's required interface methods.

We will illustrate the Executor abstraction behavior by examining the Train Executor, which realizes training of some model on some data (see Fig. 2). This Executor requires the Model and the Dataset Loader abstractions to implement the *train_function* and *get_dataset* methods respectively. During the experiment run the Executor calls the Dataset Loader's *get_dataset* method, receives the required data, passes it to the Model and calls its *train_function* method. Methods are called with user specified parameters. An experiment run with this Executor results in a newly trained Model that gets uploaded into the registry. The trained model can then be reused in further experiments.

4.2.2 Model

A Model abstraction consists of an ML model defined with any Python ML framework (such as PyTorch or TensorFlow) and a lightweight wrapper with typed interfaces required to interact with target Executors (such as the *train_function* method which trains the model, see Fig. 3), which is imported from the Python SDK. Creating a MLManagement Model abstraction is trivial. A Model is suitable for usage with any Executor which requires some subset of interfaces realized within this Model (see Fig. 4).

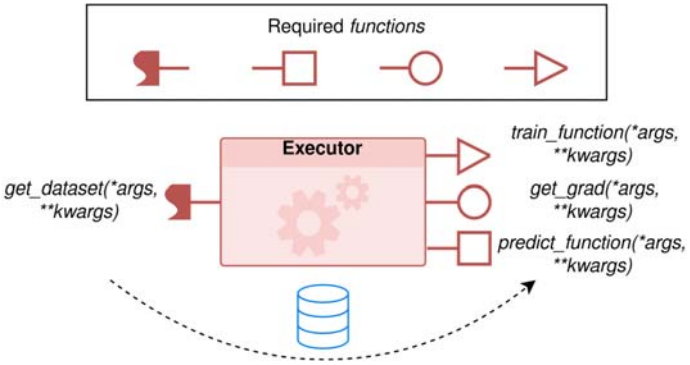

Fig. 2. Executor operates other abstractions via interfaces, allowing for reusing abstractions that implement these interfaces.

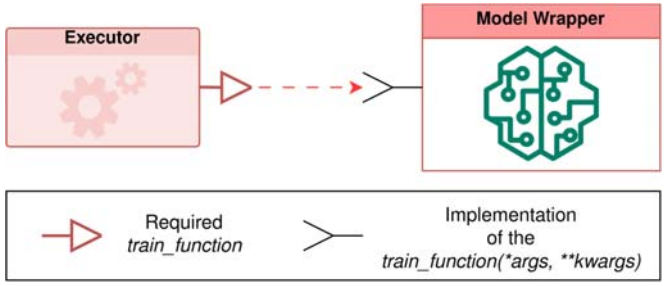

Fig. 3. Model realizes interfaces needed by a particular Executor.

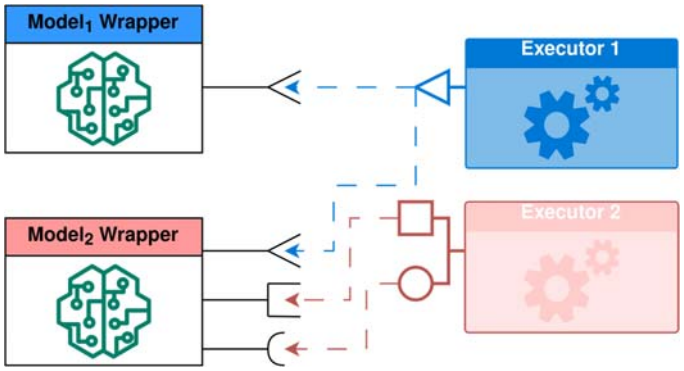

Fig. 4. Model 1 is only suitable for Executor 1, while Model 2 is suitable for both Executor 1 and Executor 2 due to its implemented interfaces.

4.2.3 Dataset Loader

A Dataset Loader abstraction is used for transforming data into the form that is required by some model. It standardizes the process of data transformation as part of the MLManagement platform. Realization of various interfaces in a lightweight wrapper allows for reusing one Dataset Loader in multiple experiments with various Executors and Models (see Fig. 5).

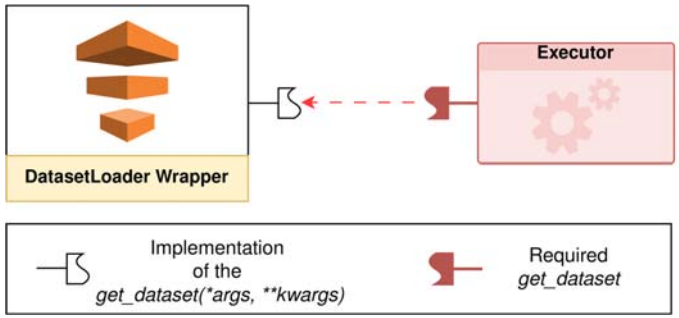

Fig. 5. Dataset Loader realizes interfaces needed by a particular Executor.

4.2.4 Experiment construction

The MLManagement platform allows for experiment creation in just a few clicks, taking advantage of the reusable building block abstractions. In order to create an experiment on the platform, the user selects the appropriate Executor, Model and Dataset Loader abstractions via the platform UI or the Python SDK. Models and Dataset Loaders that are not suitable for the selected Executor are automatically filtered out. Then, the user specifies the desired hardware resources such as GPUs, as well as the arguments required by the selected Executor, which are then passed to the downstream abstractions' interfaces.

If an experiment requires abstractions not already present on the platform, the user is required to define them via lightweight wrappers and upload them onto the platform via Python SDK.

The experiment jobs that are created in the MLManagement platform are executed in isolated Docker containers, while saving all the user specified parameters for all the abstractions (see Fig. 6). This allows for job reproducibility, as well as avoiding conflicts of requirements.

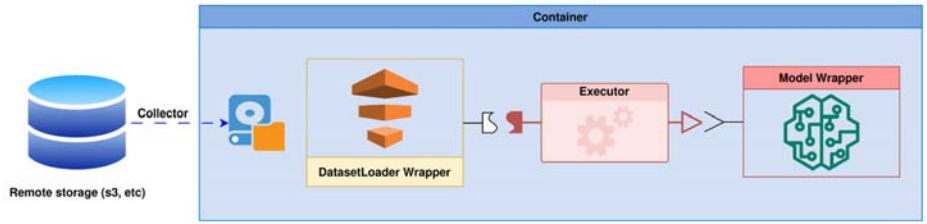

Fig. 6. Construction of automated experiments via combining reusable abstractions in an isolated environment.

After an experiment is created, the user can monitor its status, resource utilization, logs and metrics via the platform UI or the Python SDK in real time.

In conclusion, our platform allows for speedy creation of automated experiments via agile reusable abstractions, allowing for better collaboration, improved codebase with less code duplication and faster ML research.

5. Evaluation

For a quantitative evaluation we chose a multi-user model registry interaction simulation. Each of ten ML engineers attempts a simple experiment of finetuning a model with a frozen BERT backbone and a small trainable head using an MLOps tool on a common computational cluster. Finetuning requires ten runs, each resulting in a saved model. The backbone uses a lot of disk space, and thus, if it gets copied for each experiment by each user, it can quickly fill up storage. In contrast, access control and a unified model registry allows for implicitly sharing the same backbone across multiple experiments and users.

We carry out quantitative disk usage comparison to evaluate the benefits of access control and built-in data storage with artifact compression implemented in our platform, as compared to MLFlow (with different model storage formats) and DVC. We also assess the influence of this feature on basic model actions' latency, namely, time to upload a new model and time to download an existing model. The results are presented in Table 2.

The comparison demonstrates that our tool uses the disk space much more efficiently due to implemented collaboration features and artifact compression, but is slightly slower due to additional logic.

Table 2. Disk utilization in a multi-user ML experimentation setting.

Tool name	Disk utilization, MB ↓	Upload time, sec. ↓	Download time, sec. ↓
MLFlow (Pytorch)	88621.9	25 ± 2	14.2 ± 0.7
MLFlow (Pyfunc)	44432.2	11.9 ± 0.2	16.6 ± 0.7
DVC	4790.6	$\mathbf{1.9 \pm 1.1}$	$\mathbf{7.8 \pm 0.3}$
MLManagement (ours)	**478.9**	10 ± 3	9 ± 1

6. Conclusion and future work

This paper presented MLManagement, the MLOps platform which addresses limited collaboration ability and limited experiment code reusability present in current open-source tools by employing the reusable building block abstractions, ABAC and hardware resources allocation. Our next step will be to perform a case study of tool impact on time to market.

The results were obtained using the equipment of the Shared Research Facility «Shared Research Center of the Ivannikov Institute for System Programming of the Russian Academy of Sciences (SRC ISP RAS).

References

[1]. Ruf P., Madan M., Reich C., Ould-Abdeslam D. Demystifying MLOps and presenting a recipe for the selection of open-source tools. Applied Sciences, vol. 11, no. 19, 2021, p. 886.

[2]. Renggli C., Kästner M., Kossmann D., Laesser C., Binnig C. A data quality-driven view of MLOps. IEEE Data Engineering Bulletin, vol. 44, no. 1, 2021, pp. 11-23.

[3]. Tamburri D. A. Sustainable MLOps: Trends and challenges. In: Proceedings of the 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2020, pp. 17-23.

[4]. Vanska S., Kemell K.-K., Mikkonen T., Abrahamsson P. Continuous software engineering practices in AI/ML development past the narrow lens of MLOps: Adoption challenges. e-Informatica Software Engineering Journal, vol. 18, no. 1, 2024, p. 240102.

[5]. Zaharia M., Xin R. S., Wendell P., Das T., Armbrust M., Dave A., Meng X., Rosen J., Venkataraman S., Franklin M. J., Ghodsi A., Gonzalez J., Shenker S., Stoica I. Accelerating the machine learning lifecycle with MLflow. IEEE Data Engineering Bulletin, vol. 41, no. 4, 2018, pp. 39-45.

[6]. Verma A. Insecure deserialization detection in Python. Master's thesis, San Jose State University, 2023.

[7]. Yuan E., Tong J. Attribute-based access control (ABAC) for web services. In: Proceedings of the IEEE International Conference on Web Services (ICWS'05). IEEE, 2005.

[8]. Ahmad T., Adnan M., Rafi S., Akbar M. A., Anwar A. MLOps-enabled security strategies for next-generation operational technologies. In: Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (EASE'24). Association for Computing Machinery (ACM), 2024, pp. 662-667.

[9]. Eken B., Pallewatta S., Tran N., Tosun A., Babar M. A. A multivocal review of MLOps practices, challenges and open issues. ACM Computing Surveys, 2025, in press.

[10]. Jette M. A., Wickberg T. Architecture of the SLURM workload manager. In: Workshop on Job Scheduling Strategies for Parallel Processing. Springer, 2023, pp. 3-23.

Информация об авторах / Information about authors

Максим Алексеевич РЫНДИН – научный сотрудник ИСП РАН. Сфера научных интересов: непрерывное обучение и MLOps.

Maxim Alekseevich RYNDIN – researcher at ISP RAS. Research interests: continuous learning and MLOps.

Александр Михайлович БОЙКО – младший научный сотрудник ИСП РАН. Сфера научных интересов: MLOps и LLM-in-the-Loop.

Aleksandr Mikhailovich BOIKO – junior researcher at ISP RAS. Research interests: MLOps and LLM-in-the-Loop.

Денис Олегович КУЩУК – научный сотрудник ИСП РАН. Сфера научных интересов: MLOps.

Denis Olegovich KUSHCHUK – researcher at ISP RAS. Research interests: MLOps.

Дмитрий Маратович КИРАНОВ – стажер-исследователь ИСП РАН. Сфера научных интересов: out-of-distribution detection, MLSecOps, AutoML, активное обучение.

Dmitry Maratovich KIRANOV – intern researcher at ISP RAS. Research interests: out-of-distribution detection, MLSecOps, AutoML, active learning.