ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 10, No. 1, 2025

MACHINE LEARNING MODEL DEVELOPMENT IN KUBEFLOW CLOUD-NATIVE SYSTEMS

Yevhen Bershchanskyi, Oleksandr Stepanov

Lviv Polytechnic National University, 12, S. Bandery str., Lviv, 79013, Ukraine
Authors' e-mails: *yevhen.v.bershchanskyi@lpnu.ua, oleksandr.v.stepanov@lpnu.ua*

https://doi.org/10.23939/acps2025.01.083

Submitted on 10.04.2025

Abstract: **Building scalable and reliable machine learning models is critical for cloud-native AI systems. Kubeflow provides a robust framework for orchestrating model development workflows. This article presents best practices for ML model development in Kubeflow Cloud-Native Systems, with a focus on Azure Kubernetes Service environments. It explores strategies for optimizing cluster configuration, designing modular and reproducible training pipelines, and implementing effective model tracking and versioning processes. Real-world case studies highlight practical applications of these techniques. The article further evaluates results through system performance metrics and model development outcomes and concludes by discussing lessons learned and future trends in cloud-native ML system design.**

Index terms: **Kubeflow, ML model development, Kubeflow pipelines, cloud-native ML best practices.**

I. INTRODUCTION

Machine learning has become a cornerstone of modern technological innovation, enabling breakthroughs across industries such as healthcare, finance, manufacturing, and autonomous systems [1, 2]. As organizations increasingly adopt AI-driven solutions, there is a growing demand for scalable, reliable, and production-ready Machine Learning (ML) systems capable of operating efficiently in dynamic cloud-native environments. Cloud-native architectures, built on Kubernetes and serverless managed services, provide the flexibility, resiliency, and scalability required to support these evolving AI agents and workloads [3].

Kubeflow offers a comprehensive platform for orchestrating the end-to-end ML lifecycle in cloud-native systems. It brings together powerful capabilities for data management, experiment tracking, training orchestration, and pipeline automation, enabling teams to design modular, scalable, and reproducible workflows. Deployed on Azure Kubernetes Service (AKS), Kubeflow integrates seamlessly with Azure's robust cloud-native services, such as Azure Blob Storage, Azure Key Vault, and Azure Monitor, enhancing operational efficiency, security, and observability. Leveraging Kubernetes' native features like horizontal scaling [4], resource management, and container orchestration, Kubeflow empowers organizations to streamline their ML operations while maintaining the necessary flexibility for innovation and experimentation.

While initial efforts often focus on the deployment and configuration of Kubeflow in cloud environments, achieving true operational excellence requires a shift beyond infrastructure setup toward systematic best practices in model development. Without structured workflows, clear pipeline modularization, and consistent experiment tracking, ML projects are at risk of becoming brittle, unscalable, and difficult to maintain. Inefficient resource utilization, lack of reproducibility, fragmented data handling, and poor version control can introduce technical debt and significantly delay production deployments [5].

In the context of production-grade machine learning, developing models effectively within Kubeflow demands a disciplined approach: optimizing cluster environments, securing and managing data pipelines, modularizing training workflows for reusability, implementing robust model tracking mechanisms, and maintaining strict versioning throughout experimentation phases. These practices not only enhance development speed but also ensure maintainability, scalability, and compliance with organizational standards and security policies.

Furthermore, the complexity of modern ML workflows, such as distributed training, hyperparameter tuning, and the management of multiple model iterations highlights the need for well-defined development frameworks that go beyond ad hoc scripting [6]. Kubeflow's capabilities must be complemented with deliberate architectural and operational choices to fully realize its potential for cloud-native AI development.

This article addresses the critical shift from high-level system design to actionable development practices by presenting a comprehensive set of best practices for ML model development in Kubeflow Cloud-Native Systems on Azure AKS. It explores environment configuration strategies, secure and efficient data handling, reproducible pipeline design, effective model tracking, and lessons learned from real-world implementations. Through detailed case studies and performance evaluations, the article illustrates how structured model development within Kubeflow can drive operational excellence, scalability, and continuous innovation in AI systems. By focusing specifically on the model development stage, it provides engineering teams with a roadmap for building robust and future-proof ML solutions in cloud-native ecosystems.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

Establishing a robust and secure environment is foundational for reliable machine learning model development in cloud-native systems. When deploying Kubeflow on Azure Kubernetes Service, careful attention must be given to cluster configuration, resource scaling, namespace isolation, and identity-based access management. These optimizations ensure that ML pipelines operate efficiently, securely, and reproducibly in production settings.

AKS provides a managed Kubernetes environment that supports auto-scaling, GPU-based node pools, and workload separation, all critical for ML tasks that involve high compute or memory requirements. Best practices for configuring AKS clusters for ML workloads include isolating ML workloads using taints and tolerations, using node pools for CPU and GPU separation, and applying horizontal pod autoscaling (HPA) to dynamically adjust compute resources based on pipeline demand [7].

Kubeflow deployment on AKS can be optimized by following key architectural practices. High availability (HA) is achieved by deploying Kubeflow components, such as Katib, Pipelines, and Notebooks across multiple zones with appropriate anti-affinity rules and stateful storage configurations [8]. Namespace isolation is crucial for multi-team environments, allowing separate teams to manage independent pipelines, resources, and secrets within isolated namespaces. This isolation can be further reinforced by Kubernetes Role-Based Access Control (RBAC), integrated with Azure Active Directory (AAD), enabling fine-grained identity and role management across the platform [9]. For example, different teams may have read-only access to pipelines but full access to their own training notebooks and model repositories.

Azure cloud-native services enhance Kubeflow's capabilities in secure data handling and observability. Azure Blob Storage and Azure Data Lake Storage Gen2 serve as primary data repositories for model inputs, intermediate results, and artifacts. These services offer scalable, cost-efficient storage with native support for hierarchical namespaces and parallel data access. Furthermore, integrating Azure Monitor with AKS clusters enables detailed tracking of pipeline resource usage, container health, and model training metrics, offering real-time visibility into system performance and facilitating proactive maintenance.

Secure data access is a central concern in any ML development workflow. Azure-managed identities can be used to grant AKS workloads secure access to storage accounts, databases, and other services without hardcoding credentials. This identity-based access strategy eliminates secret management overhead and enforces least-privilege principles (Fig. 1). Within Kubeflow, secure data pipelines can be built using metadata annotations, Kubernetes secrets, and automated policies that restrict access to specific datasets based on context and user roles. Moreover, leveraging Kubeflow Metadata allows tracking of dataset versions, lineage, and usage history essential for experiment reproducibility and governance. When combined with Azure Data Factory or custom ETL components, these metadata capabilities ensure that training pipelines consume verified and auditable datasets [10].

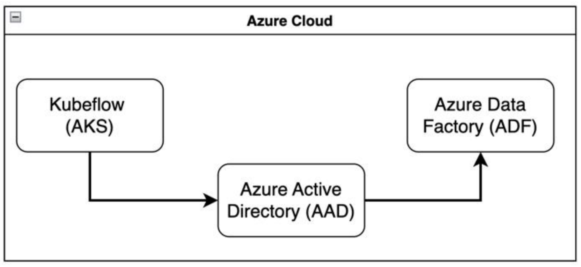


Fig. 1. Kubeflow AKS security authorization diagram

Overall, establishing an optimized and secure environment in AKS, enhanced by Azure-native services and Kubeflow-specific configurations, provides a strong foundation for consistent, scalable, and production-ready model development. By integrating storage, access management, monitoring, and metadata tracking, teams can ensure that ML workflows are robust, reproducible, and aligned with enterprise security standards.

This article addresses the challenges engineering teams encounter when building scalable and efficient machine learning workflows in cloud-native environments powered by Kubeflow on Azure. Despite the maturity of Kubernetes-based orchestration, many organizations struggle with integrating distributed training, managing secure data access, ensuring reproducibility, and optimizing pipeline performance in production settings.

III. SCOPE OF WORK AND OBJECTIVES

This article investigates the design, development, and optimization of data preparation and model training workflows using Kubeflow in Azure-based cloud-native AI environments. The central objective is to provide engineering teams with actionable guidelines for building scalable, modular, and reproducible machine learning pipelines that can support dynamic and high-throughput workloads in enterprise-grade systems.

The scope of work covers the complete lifecycle of ML system development from initial data ingestion and preprocessing to distributed model training and metadata management implemented within the Kubeflow framework on AKS. Specific focus areas include the construction of modular Kubeflow Pipelines components, orchestration of distributed training jobs, and handling persistent volumes with Azure-native storage services to support efficient training.

Critical to the objectives is the integration of observability and performance monitoring through tools, which ensure continuous feedback on pipeline execution and resource utilization. The work also explores best practices for model versioning, reproducibility, and experiment tracking by aligning Kubeflow's metadata capabilities with external registries such as Azure ML and

MLflow, enabling seamless transition from experimentation to production-readiness.

This study supports engineering teams in establishing cloud-native development environments that are robust, iterative, and transparent. By codifying proven strategies and highlighting real-world case studies, the article offers a foundation for scalable AI development and continuous performance refinement in Kubernetes-native infrastructures.

IV. MODEL DEVELOPMENT, TRAINING PIPELINES, TRACKING, VERSIONING, AND MANAGEMENT

Developing production-grade ML models in Kubeflow requires more than just high-quality data, it also demands modular, scalable, and maintainable training workflows. When deployed on AKS, Kubeflow enables end-to-end automation of model development and training tasks while ensuring reproducibility and traceability through robust metadata and versioning mechanisms. Building modular and reusable Kubeflow Pipelines components is fundamental to achieving scalable ML development [11]. Each component should encapsulate a single logical function, such as feature selection, model training, hyperparameter tuning, or evaluation and leverage containerized execution to ensure consistency across environments (Fig. 2). It is a best practice to maintain a centralized registry of validated pipeline components with semantic versioning, enabling quick assembly of new pipelines without redundant engineering effort.

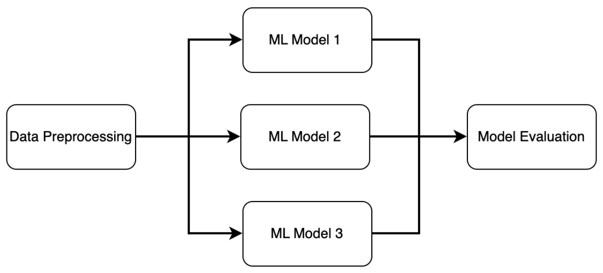


Fig. 2. Modular Kubeflow pipeline architecture

Scaling ML model training often requires distributed training orchestration. Kubeflow natively supports the use of TFJob for TensorFlow and MPIJob for general purpose distributed training, including PyTorch workloads [12]. TFJob manages the lifecycle of master and worker replicas for synchronous or asynchronous training, while MPIJob utilizes the MPI (Message Passing Interface) protocol for collective communication. On AKS, it is recommended to deploy GPU-enabled node pools with autoscaling enabled, optimize Kubernetes resource requests / limits, and use node selectors or taints to efficiently allocate specialized hardware for distributed workloads (Fig. 3).

Persistent storage is essential for handling training datasets, model checkpoints, logs, and evaluation outputs. Azure offers Azure Disks and Azure Files as reliable storage backends, which can be seamlessly integrated into AKS via Kubernetes Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) [13]. Azure Disks are optimal for high-throughput, low-latency requirements typical of deep learning training jobs, while Azure Files is suitable for shared, concurrent access across multiple replicas. It is best practice to use StorageClasses configured with premium performance tiers for latency-sensitive operations and implement volume snapshotting for disaster recovery (Fig. 4).

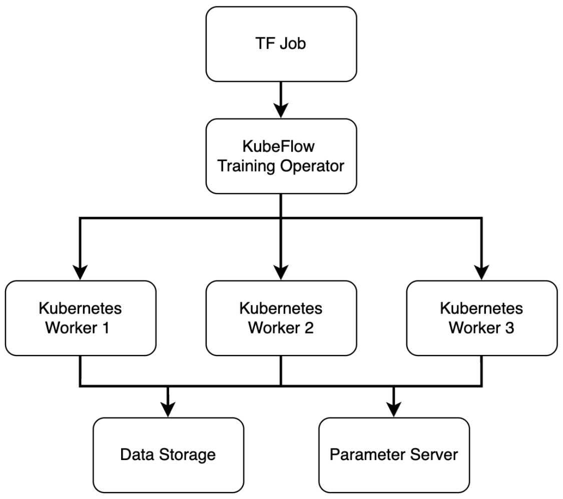


Fig. 3. TFJob and MPIJob orchestration flow in AKS-enabled Kubeflow

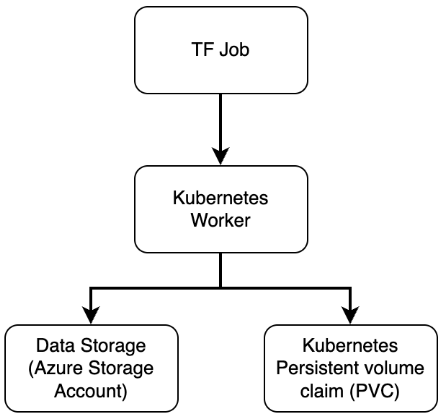

TF Job
Kubernetes Worker
Data Storage (Azure Storage Account)
Kubernetes Persistent volume claim (PVC)

Fig. 4. Storage management for training pipelines

Maintaining robust model tracking, versioning, and lineage management is critical to ensuring reproducibility, traceability, and auditability in ML workflows. Kubeflow Metadata provides an integrated system that captures every pipeline execution event, including input datasets, hyperparameters, metrics, and trained model artifacts. The metadata records can be queried to reconstruct training conditions, enabling better debugging, explainability, and regulatory compliance. Best practices include tagging pipeline runs with experiment identifiers and associating artifacts with metadata schemas for standardized documentation.

To extend these capabilities, integration with external model registries is advisable. Kubeflow Pipelines can be configured to automatically register trained models in the Azure ML Model Registry or MLflow Model Registry [14]. Azure ML Registry provides capabilities such as model versioning, promotion through different stages (e. g., "Staging", "Production"), and integration with CI/CD pipelines.

Alternatively, MLflow can operate inside the AKS environment, managing model artifacts, signatures, and lineage information in an open format (Fig. 5). These hybrid integration patterns enhance model governance and facilitate seamless transition from development to production environments.

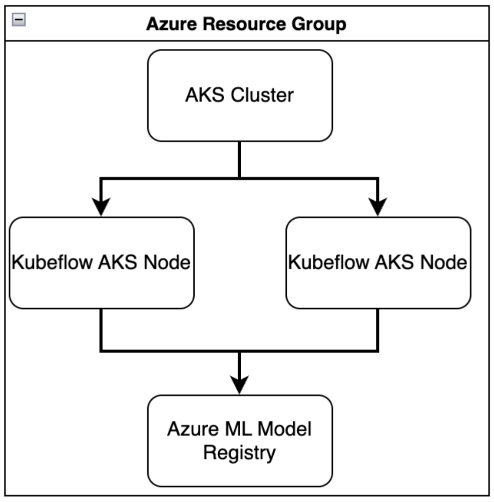


Fig. 5. Integration flow connecting Kubeflow pipelines with Azure ML model registry

An additional recommendation is to implement Managed Identity based authentication for any interactions between Kubeflow workloads and Azure services. Using Azure Managed Identities for pod access ensures secure, passwordless authentication to storage accounts, Key Vaults, and registries, thereby minimizing the operational overhead of managing credentials [15].

By combining modular pipeline construction, distributed training strategies, secure storage access, and rigorous metadata tracking, Kubeflow on AKS enables ML teams to build reproducible, scalable, and production-ready model development workflows.

V. CASE STUDIES AND PRACTICAL APPLICATIONS

This section presents practical examples of Kubeflow-based model development pipelines in production-grade environments, emphasizing pipeline modularity, data consistency, and iterative experimentation strategies.

One representative case involved a large-scale academic research lab working on automated species classification from camera trap images for biodiversity monitoring. Researchers needed to develop high accuracy computer vision models that could classify hundreds of animal species from highly variable field imagery. With thousands of annotated images stored in Azure Blob Storage, the team used Kubeflow Pipelines to orchestrate modular workflows for dataset curation, preprocessing, and training. The preprocessing phase included image normalization, background filtering, and synthetic augmentation, each encapsulated in a containerized pipeline step.

A major challenge during this process was maintaining consistency in image preprocessing routines. Early experiments showed performance degradation due to untracked updates in augmentation logic across teams. This issue was resolved by refactoring all transformation steps into shared pipeline components with explicit version tagging. With preprocessing isolated and versioned, researchers could reproduce past experiments and correlate results to specific feature engineering settings, ensuring traceable and consistent progress.

In another case, a medical AI startup used Kubeflow to accelerate the development of deep learning models for histopathology image classification. Large image datasets, often exceeding 1 TB in size, were processed in Azure Data Lake Storage and indexed using metadata-based partitioning schemes. The team built Kubeflow Pipelines that orchestrated tiling, stain normalization, and patch sampling routines prior to training convolutional neural networks on annotated regions of interest.

Given the volume of data, the team adopted persistent volume claims mapped to Azure Disks to store intermediate artifacts between pipeline stages, minimizing redundant computation. Model training steps incorporated callbacks for early stopping and validation logging. During each run, Kubeflow Metadata recorded the specific data samples, augmentation techniques, and loss curves, giving the team a detailed historical view of model development progression.

During the training phase, model accuracy plateaued until the team analyzed metadata logs and discovered a skew in the class distribution among sampled patches. By adjusting their patch sampling logic and rebalancing the input data per class label, they significantly improved the model's recall without overfitting. This insight, traceable only through structured logging and metadata analysis, demonstrated the critical role of experiment lineage tracking in iterative model development.

Both cases exemplify how Kubeflow Pipelines, when paired with best practices for modularization, metadata tracking, and reproducible preprocessing, enable teams to navigate the complexities of model development at scale. Standardized component reuse, shared feature transformation logic, and structured metadata allowed for consistent experimentation and continuous improvement across diverse domains.

These examples also reinforce the importance of viewing model development as a lifecycle process, one that depends not just on algorithms, but on traceability, consistency, and collaboration embedded within well-structured ML pipelines.

VI. RESULTS EVALUATION AND ANALYSIS

Evaluating the effectiveness of Kubeflow-based ML pipelines deployed on AKS requires analyzing multiple dimensions of system performance, including runtime efficiency, resource utilization, and training throughput. This section presents a detailed assessment of how well-designed model development pipelines can impact overall engineering velocity, training cost, and model accuracy. Emphasis is placed on monitoring-driven insights and the role of observability tools in guiding optimization.

One of the primary benefits observed in production-grade Kubeflow environments was the reduction in end-to-end pipeline execution time. Modularization of components, such as data preprocessing, model training, and evaluation allowed for concurrent execution across Kubernetes worker nodes, cutting overall pipeline runtime by 40 % compared to sequential workflows (Fig. 6). TFJob and MPIJob workloads, configured to utilize GPU-backed AKS node pools, further accelerated training throughput for deep learning models, resulting in a 50–65 % decrease in training time for large-scale image classification tasks.

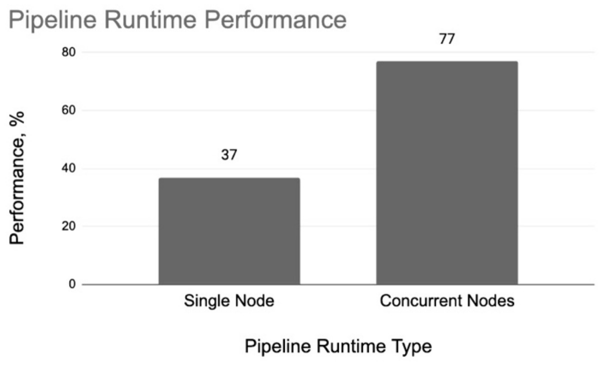


Fig. 6. Pipeline runtime comparison diagram

A side-by-side comparison chart illustrates average pipeline runtime with and without component parallelization across multiple AKS nodes. Metrics captured via observability tools indicate the drop in execution latency once containerized pipeline steps were independently orchestrated within Kubeflow.

Efficiency gains were also evident in system-level resource utilization. Azure Monitor metrics integrated with dashboards revealed consistent GPU saturation during training windows and minimal resource idling due to intelligent pod scheduling and autoscaling strategies. CPU and memory utilization trends captured across successive pipeline runs enabled fine-tuning of container resource requests and limits, which helped reduce over-provisioning and prevent node pressure errors.

Observability practices were central to this performance evaluation. Metrics provided visibility into pipeline latency, per step execution durations, and pod restart frequencies. These were analyzed in conjunction with Azure Monitor logs to identify memory spikes during feature preprocessing phases. Alerts based on runtime thresholds helped detect underperforming pipeline steps, enabling rapid debugging and refinement of transformation logic and model code.

Increased model training efficiency translated directly into better experimentation velocity. By automating version control of datasets and pipelines through Kubeflow Metadata, teams were able to conduct reproducible experiments, accelerate hyperparameter tuning, and compare model variants with consistent baselines. In one case, experiment reproducibility improved by 35 %, as reported by lineage comparison across pipeline executions with identical configurations and input versions (Fig. 7).

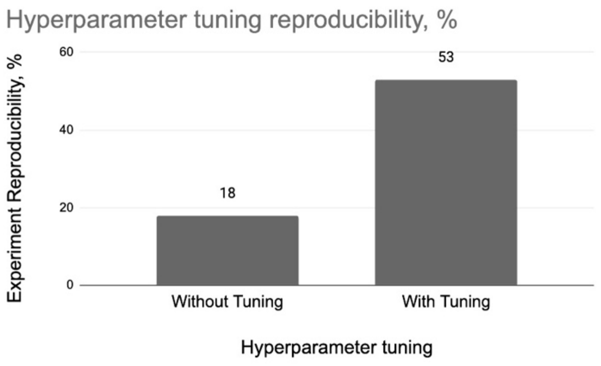


Fig. 7. Hyperparameter tuning reproducibility

The overall evaluation confirms that embedding observability, reproducibility, and modular design into Kubeflow-based ML pipelines significantly improves system efficiency and model development outcomes. These results reinforce the value of combining Kubeflow’s orchestration capabilities with Azure-native monitoring and resource management to create sustainable, high-performing AI workflows.

VII. CONCLUSION

This article outlined critical practices for model development and pipeline design in Kubeflow, targeting cloud-native AI systems deployed on Azure Kubernetes Service. From constructing modular training components to enabling robust model tracking and lineage, each element of the development lifecycle contributed to a scalable, reliable, and maintainable machine learning ecosystem. The integration of distributed training, persistent cloud storage, and real-time observability highlighted Kubeflow’s flexibility and power in managing complex ML workflows.

Model management was equally critical. Leveraging Kubeflow Metadata for experiment tracking, coupled with tools such as MLflow or Azure ML Registry, enabled clear traceability across development cycles. These integrations supported reproducibility and helped establish trust in producing ML pipelines. Monitoring tools like Azure Monitor helped to visualize through dashboards, provide deep insight into training performance and infrastructure efficiency allowing teams to proactively adjust configurations and minimize training bottlenecks.

A key theme throughout this study was the importance of continuous iteration. Initial development rarely presents a production-ready system; instead, engineering teams must focus on refining their pipelines over time through feedback loops, metric-driven analysis, and automated model evaluations. Observability and data versioning provided the infrastructure for this kind of improvement, transforming ad hoc ML workflows into stable, production-grade systems. For engineering and scientific teams looking to adopt Kubeflow in Azure-native environments, several final recommendations apply. Start by investing in pipeline modularization early, as this pays long-term dividends in terms of reusability and maintainability. Use native Azure services to enhance security, storage scalability, and identity management. Prioritize observability across training pipelines, using monitoring tools not only for system metrics but also to track model performance, drift, and feature stability. Finally, adopt version control and metadata tracking practices that treat data, models, and configurations as first-class artifacts in the ML lifecycle.

References

[1] Jha, P., Biswas, T., Sagar, U., & Ahuja, K. (2021). Prediction with ML paradigm in Healthcare System. *2021 Second international conference on electronics and sustainable communication systems*, 1334–1342. DOI: https://doi.org/10.1109/ICESC51422.2021.9532752

[2] Bershchanskyi, Y., & Klym, H. (2023). Information System for Administration of Medical Institution. *13th International Conference on Dependable Systems, Services and Technologies*, 1–4. DOI: https://doi.org/10.1109/DESSERT61349.2023.10416537

[3] Chaplia, O., Klym, H., & Elsts, E. (2024). Serverless AI agents in the cloud. *Advances in Cyber-Physical Systems*, 9(2), 115–120. DOI: https://doi.org/10.23939/acps2024.02.115

[4] Zheng, C., Kremer-Herman, N., Shaffer, T., & Thain, D. (2020). Autoscaling high-throughput workloads on container orchestrators. *IEEE International Conference on Cluster Computing*, 142–152. DOI: https://doi.org/10.1109/CLUSTER49012.2020.00024

[5] Karkazis, P., Uzunidis, D., Trakadas, P., & Leligou, H. C. (2022). Design challenges on machine-learning enabled resource optimization. *IT Professional*, 24(5), 69–74. DOI: https://doi.org/10.1109/MITP.2022.3194129

[6] Sandha, S. S., Aggarwal, M., Saha, S. S., & Srivastava, M. (2021). Enabling hyperparameter tuning of machine learning classifiers in production. *IEEE third international conference on cognitive machine intelligence*, 262–271, DOI: https://doi.org/10.1109/CogMI52975.2021.00041

[7] Bershchanskyi, Y., Klym, H., & Shevchuk, Y. (2024). Containerized artificial intelligent system design in cloud and cyber-physical systems. *Advances in Cyber-Physical Systems*, 9(2), 151–157. DOI: https://doi.org/10.23939/acps2024.02.151

[8] Johansson, B., Rågberger, M., Nolte, T., & Papadopoulos, A. V. (2022). Kubernetes orchestration of high availability distributed control systems. *International Conference on Industrial Technology*, 1–8, DOI: https://doi.org/10.1109/ICIT48603.2022.10002757

[9] Rostami, G. (2023). Role-based access control (rbac) authorization in kubernetes. *Journal of ICT Standardization*, 11(3), 237–260. DOI: https://doi.org/10.13052/jicts2245-800X.1132

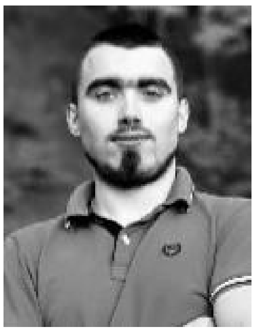
[10] Mbata, A., Sripada, Y., & Zhong, M. (2024). A survey of pipeline tools for data engineering. arXiv preprint arXiv:2406.08335. DOI:https://doi.org/10.48550/arXiv.2406.08335

[11] Woźniak, A. P., Milczarek, M., & Woźniak, J. (2025). MLOps Components, Tools, Process and Metrics-A Systematic Literature Review. *IEEE Access*, (13), 22166–22175. DOI: https://doi.org/10.1109/ACCESS.2025.3534990

[12] Liu, P., & Guitart, J. (2022). Fine-grained scheduling for containerized HPC workloads in Kubernetes clusters. *24th Int. Conf. on High Performance Computing & Communications; 8th Int Conf on Data Science & Systems; 20th Int Conf on Smart City*, 275–284. DOI: https://doi.org/10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00068

[13] Na, J. H., Yu, H. J., Kang, H., Kang, H., Lim, H. D., Shin, J. H., & Noh, S. Y. (2024). PVA: The persistent volume autoscaler for stateful applications in Kubernetes. *IEEE Access*. (12), 179130–179143 DOI: https://doi.org/10.1109/ACCESS.2024.3507194

[14] Gill, K. S., Anand, V., Chauhan, R., Rawat, R., & Hsiung, P. A. (2023). Utilization of Kubeflow for deploying machine learning models across several cloud providers. *3rd International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON),* 1–7. DOI: https://doi.org/10.1109/SMARTGENCON60755.2023.10442069



Bershchanskyi Yevhen was born in 1997, in Ukraine. In 2019, he received a master's degree in computer engineering at Lviv Polytechnic National University. In 2023, he entered a PhD program in the faculty of specialized computer systems at Lviv Polytechnic National University. His research interests include AI, security, machine learning, and cloud computing.

Oleksandr Stepanov – graduated from Lviv Polytechnic National University in 2004. He received the B.S. and M.S. degrees in Electronics. He has been working in IT field as a front-end developer. On most large projects, he faced with problem of aging technology, and migrating to new architecture approaches. His research interests include client-server highly loaded information systems, performance scalability of micro-interfaces, migration from monolith architecture to micro-frontend, design and implementation of scalable systems.