

Article

Scalable MLOps Pipeline with Complexity-Driven Model Selection Using Microservices

Oleh Pitsun *  and Myroslav Shymchuk

Department of Computer Engineering, West Ukrainian National University, Lvivska, 11, 46003 Ternopil, Ukraine; miroslavshimchuk@gmail.com

* Correspondence: o.pitsun@wunu.edu.ua

Abstract

The increasing complexity of integrating modern convolutional neural networks into software systems imposes significant computational demands on machine learning infrastructures. Existing MLOps systems lack mechanisms for dynamic model selection based on dataset complexity, leading to inefficient resource utilization and limited scalability under high-load conditions. This study employs convolutional neural network-based machine learning algorithms for image classification and ensemble methods for quantitative feature classification. The paper presents a self-optimizing machine learning pipeline that integrates a microservices-based architecture with a formal process for estimating image complexity and an optimization-based model selection strategy. The proposed methodology is based on designing an adaptive microservice-based ML pipeline that dynamically reconfigures its computation graph at runtime. The results confirm the effectiveness of the proposed approach for building resilient and high-performance distributed systems. The mechanism proposed in this work enables the adaptive use of modern deep learning algorithms, leading to improved result quality. A comparative analysis with existing approaches demonstrates superiority in model selection complexity, pipeline overhead, and scalability. The outcome of the proposed mechanism is an adaptive algorithm selection process based on bias-related parameters, enabling the selection of the most suitable module for data processing.

Keywords: microservices; distributed systems; cloud-native architecture; service orchestration; scalability



Academic Editor: Pedro Antonio Gutiérrez

Received: 7 December 2025

Revised: 2 January 2026

Accepted: 5 January 2026

Published: 7 January 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The development of artificial intelligence tools across various domains of human activity necessitates designing new software architectures that account for the specific requirements of machine learning components, including models, datasets, libraries, and computational resources. Monolithic ML systems are increasingly unable to cope with the exponential growth in model complexity, ranging from simple architectures to transformer-based models. Furthermore, the cost of errors or failures in complex systems continues to rise, potentially compromising overall system reliability.

The rapid deployment of deep learning models in production environments has significantly increased the computational and operational complexity of modern ML systems. In the development of large-scale systems, it is essential to anticipate all required components and modules; therefore, pipeline design has become a key stage in the architectural development process. With the evolution of cloud computing, substantial attention has been

devoted to DevOps pipelines. Similarly, the widespread adoption of artificial intelligence, intense learning, has led to a growing focus on MLOps and machine learning pipelines.

In software–hardware systems that leverage cloud computing and distributed training, the ability to automatically select an appropriate model architecture based on task complexity and available computational resources is critical. Optimizing ML pipelines is increasingly essential, as training large-scale models can consume significant computational resources and result in substantial financial costs. The microservice paradigm allows you to divide a complex ML system into independent components. The microservice architecture enables horizontal scaling of individual services, optimal distribution of GPU/CPU resources, and flexible orchestration using Kubernetes (v1.32.11)/Kubeflow (version 1.10). Unlike monolithic ML platforms, microservices allow you to easily implement new models and algorithms without disrupting the operation of the entire system.

Despite significant progress in MLOps tools, modern ML systems have several shortcomings, namely that many existing pipelines require manual selection of convolutional neural network architectures based on data complexity. In addition, many existing pipelines focus on calculating quantitative indicators for specific objects in numerical data. When working with media content, manual parameter adjustments are required. Special attention is paid to processing datasets, namely the problematic storage of large datasets and their transfer between services, as this incurs high overhead.

Existing MLOps frameworks primarily focus on model deployment, monitoring, and reproducibility, but provide limited support for adaptive decision-making at runtime. As a result, production systems often suffer from inefficient resource utilization, increased inference latency, or unnecessary over-provisioning when high-capacity models are used for low-complexity data. Automated model optimization approaches, such as neural architecture search, address model selection during training but incur substantial computational cost and do not support runtime adaptation to changing data characteristics.

This work addresses the gap by proposing an adaptive ML pipeline that dynamically reconfigures its computation graph and selects an appropriate deep learning model at runtime based on formal data-complexity criteria and available system resources.

The problem that arises in modern conditions is the need to create a universal, adaptive ML pipeline that automatically evaluates dataset characteristics and selects the most effective deep learning model based on data complexity, sample size, available hardware resources, and accuracy requirements.

This work aims to develop an ML pipeline with flexible elements that enable the most efficient selection of neural network models of different types based on the input dataset. Versatility is ensured by automatically assessing the complexity of the computer vision task based on the dataset parameters, image type, and image characteristics. Automatic model selection is ensured by prior training and enables the most efficient classification of data, accounting for accuracy and quality.

The scientific novelty lies in the proposed architecture of an adaptive ML pipeline that dynamically rebuilds its computation graph based on data characteristics, load level, and hardware resources.

Thus, the creation of an adaptive microservice pipeline that automatically selects a deep learning model based on formal criteria for data complexity and available resources is a promising direction for developing modern ML systems.

Contribution:

1. A method for selecting the optimal convolutional neural network model for image classification, depending on the input characteristics of the dataset within the framework of the developed ML pipeline, is proposed, which allows for optimal use of resources in the quality–processing time ratio.

2. A universal adaptive ML pipeline is developed, which automatically selects a model depending on the complexity of the data and available resources and includes a microservices approach, which allows for increased reliability, scalability, modularity, distribution, and support for deep learning in a cloud environment.
3. A quantitative characteristics classification module based on the ensemble method is developed within the framework of the proposed ML pipeline. Using the optimizer allows you to select the optimal combination of classifiers, enabling higher accuracy and reduced processing time.
4. The advantages of the proposed approach over existing MLOps solutions are shown, in particular in the context of highly loaded systems, large datasets, and deep neural networks.

2. Literature Review

In [1], the conceptual and technological basis for creating a new generation of intelligent educational platforms based on microservice architecture, cloud technologies, and machine learning is described. In [2], a modular ML pipeline is proposed to enable the creation of adaptive, scalable artificial intelligence systems. The pipeline includes network traffic analysis, standardization, feature engineering, training several basic models, and combining them into a stack ensemble with a meta-learner. ML pipelines are widely used in various areas of human life, including ecology. In [3], the IAQ-STL-ML predictive pipeline is proposed, which combines STL decomposition with meta-learning to improve short-term forecasting accuracy and overcome forecast delays. In [4], the MLSToolbox Code Generator is presented, which allows you to graphically define ML pipelines and automatically generate high-quality Python code that adheres to key software engineering principles. ML pipelines are also used for gene analysis; in particular, work [5] presents the possibilities of analyzing quantitative indicators using machine learning tools. Pipelines are also used for LLM models; in particular, work [6] proposes DSPy, a systematic software model for the automatic training and optimization of LM pipelines.

The authors of [7] describe the basic principles for implementing an ML pipeline in O-RAN, analyze the challenges of latency and reliability across two deployment scenarios. A significant part of the research focuses on the application of pipelines in medicine, as this area is characterized by a wide range of input data, including images, videos, and numerical data. The topic of classifiers and medical data analysis is addressed in [8]. The authors of [9] pay attention to the role of artifacts in machine learning. They find that proper artifact removal and outlier management are critical to improving the accuracy of ML models on MER data for STN localization. An essential aspect of any system is the use of ready-made models for data classification. In particular, the authors of [10] analyzed YOLOv10 for automated organoid segmentation and demonstrated the benefits of a hybrid pipeline with ResNet50 and ML classifiers for highly accurate and reproducible morphological studies. In [11], the authors paid significant attention to data preprocessing, proposed an approach to preprocessing and feature engineering that makes these stages a key part of a scalable, interpretable, and robust Data Mining process, along with practical recommendations on leakage control.

New deep learning technologies offer many opportunities, but they also require substantial resources and increase complexity. In this case, when developing and designing new systems, particularly for large ones, great attention is paid to microservice approaches. In [12], the evolution of approaches to software development from traditional models, such as Waterfall, to modern, flexible approaches. Article [13] proposes a high-load architecture for the classification and segmentation of biomedical images and highlights its key

components. Additionally, a comparative analysis with analogues is conducted to assess advantages against key criteria.

Articles [14–17] analyze the main technologies for implementing a microservice approach to designing new high-load systems. The article [18] discusses the fundamental principles of microservices—service decomposition, communication, data management, and essential design patterns, such as API Gateway, Circuit Breaker, and Service Discovery, which help solve typical problems, in particular, service interaction, fault tolerance, and gradual migration from legacy systems. The paper [19] provides information on the integration of Zero Trust, which significantly enhances the security of microservice ecosystems, especially in access control. A proof-of-concept experiment confirms the effectiveness of the proposed authentication mechanism. The article [20] presents a systematic review of modular monolithic architecture and summarizes scientific data on its application in cloud systems. In [21], a new adaptive security strategy is proposed that uses queuing theory to optimize security parameters in cloud-native microservice architectures dynamically. The method allows for a simultaneous increase in system security while maintaining the required indicators.

An important aspect is the migration from monolithic architecture to microservice architecture. The study [22] presents Mono2Micro, a systematic framework for automatic migration of monolithic systems to microservice architecture. This framework combines the analysis of database structures, service decomposition, and communication design with machine learning algorithms to determine microservice boundaries and optimize their interactions accurately. When developing pipelines, it is essential to adhere to modern approaches to organizing work, distributing functionality, and establishing interaction. The works [23–26] provide examples of applying MLOps and DevOps approaches to developing complex computer systems on cloud platforms. The Agentic AIOps approach proposed in [27] integrates agent AI to autonomously detect, classify, and eliminate incidents, ensuring high system availability, performance, and stability.

Another important aspect is the use of ensemble learning algorithms to achieve optimal classification performance. In [28], a method for building ensembles from large libraries of models, each trained with different algorithms and parameters, is proposed. Using forward stepwise selection, models are added to the ensemble that maximize the selected metric. The authors [29] show that its efficiency varies across model qualities at different points in the feature space and propose a generalization—Bayesian hierarchical stacking—in which model weights vary with the data and are determined through Bayesian inference.

Thus, the analysis of literary sources confirms the relevance of the task and the need to develop pipelines for machine learning systems capable of selecting optimal models and algorithms to increase productivity.

3. Materials and Methods

3.1. Generalized Proposed Pipeline

A microservice architecture was selected as the development approach instead of a monolithic architecture. This choice enables the distribution of system logic across independent modules and prevents excessive system overload when integrating multiple machine learning algorithms and datasets. Containerization mechanisms were employed to support the operation and deployment of individual modules.

The tools used in this development are aligned with established DevOps and MLOps practices; however, they incorporate unique elements specifically designed to address the problem of optimal model architecture selection. The developed modules support both frontend and backend components, thereby enhancing the overall system's performance

and versatility. The generalized structure of the proposed architecture is illustrated in Figure 1.

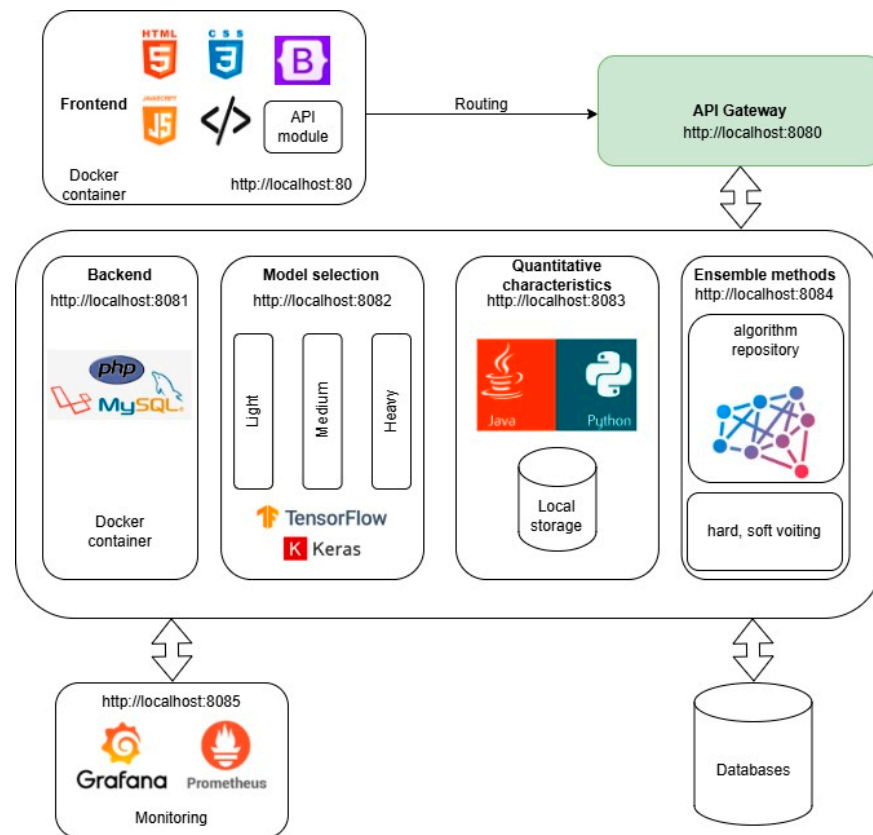


Figure 1. Generalized structure of the proposed architecture.

1. The frontend module is responsible for displaying the main module settings in a graphical form on a website with a database. This approach will allow you to develop functionality for convenient interaction with the service without requiring significant programming knowledge. The main component of this module is the use of Twitter Bootstrap as a front-end framework to ensure the graphical interface's adaptability. The use of Docker (version 29.1.3) containers is a standard approach to ensure containerization. Additionally, the module includes separate functionality to simplify working with API Gateway.
2. API Gateway is a small but very significant module that is located on port 8080 and is designed to act as an intermediary between the frontend and parts of microservices.
3. The server part of the code, which ensures the functioning of the site system, is implemented using the Laravel framework and the MySQL (version 8.4) database. The primary purpose of this approach is to store user information, conduct research, communicate using a client-server architecture, and work with web applications.
4. The model selection module for classification and segmentation is key in this architecture, which allows you to adapt the input data set to a specific type of network to obtain the optimal result in terms of classification quality, execution time, and resource usage. Three main categories of models are distinguished, namely easy, medium, and heavy.
5. The quantitative characteristics module is designed to perform segmentation tasks and calculate quantitative characteristics of micro-objects, such as area, perimeter, circumference, axis length, etc. This module allows selection of the necessary parameters for local storage in a database for further classification or clustering.

6. The ensemble-based classification module is implemented using more than 10 data classification algorithms and is designed to select the most optimal combinations for voting in soft or hard mode.
7. Prometheus (version 3.8.1) and Grafana (version 12.3.0) technologies are used to monitor the system. The module is implemented as a separate microservice.
8. The database is an essential component and is implemented as a separate storage, and it combines mechanisms for storing media objects, images, and text data.

3.2. Self-Optimizing ML Pipeline

The Self-Optimizing ML Pipeline is designed to automatically adapt the processes for training and deploying deep learning models. This pipeline combines a microservice architecture with automated monitoring and elements of automatic model integration. This approach allows you to ensure scalability, increase processing speed, and adaptability.

The developed Self-Optimizing ML Pipeline automatically adapts to the type of input data, the task of execution, for example, classification, segmentation, generation, bandwidth of communication channels, and access time to the data store.

The goal is to automatically adjust the operation parameters based on the input data, thereby providing the most productive approach to completing the task.

This pipeline has several structural subdivisions. Data Intake Service analyzes the properties of input files and determines the level of complexity. Adaptive Pre-processing Service is engaged in selecting parameters to change image size and, if necessary, the intensity of data augmentation. This block also selects pipeline branches for different formats. The following operating modes are distinguished in this pipeline:

1. Light;
2. Balanced;
3. Heavy.

Model Selection & Routing Service is designed to select the type of neural network and architecture depending on the level of complexity. It is based on indicators such as model runtime and additional parameters. This service also allows you to store metadata. That is, information about experiments, datasets, architectures, models, and automatic identification of outdated models through performance metrics.

Pipeline Orchestrator is a central component that builds a graph of computational stages, chooses the optimal route, and changes the model, configuration, and parameters during execution.

The pipeline is optimized according to the characteristics of the input data. For example, when the dataset consists of small JPEG images, a lightweight processing pipeline is selected, whereas Whole Slide Images require a fully tiled pipeline.

The optimization and adaptive learning service is designed to enable the automatic selection of hyperparameters, the choice of an optimal network architecture, and the selection of optimizers and loss functions tailored to the specific task.

In cases of high data complexity, decisions may include reducing image dimensionality, switching between model architectures (e.g., from ResNet50 to EfficientNet-B2), increasing the batch size, or enabling caching mechanisms. A key aspect of the proposed approach is the automatic switching between models of different complexity based on the current system load.

The monitoring and self-diagnostics module supports inference performance monitoring, tracking of training metrics (e.g., loss, accuracy, IoU, Dice), and the automatic detection and analysis of deviations and anomalies.

Microservice decomposition is implemented as modules, as shown in Figure 2.

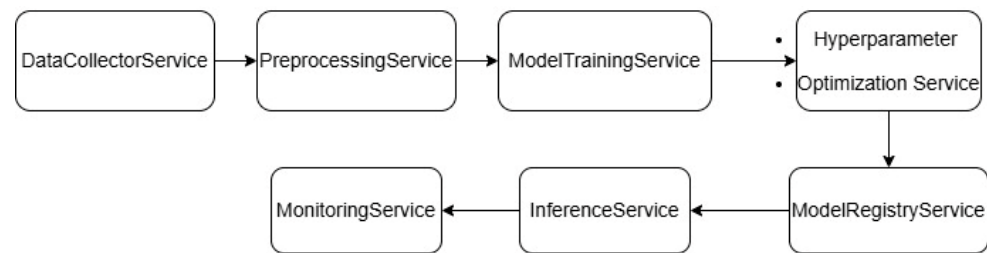


Figure 2. Microservice architecture elements.

This approach allows you to independently scale services, deploy different model versions, and allocate computing resources as needed. Division into 3 categories allows for optimal load distribution, and the router, which acts as a task distributor, is decisive in this context. The pipeline architecture is shown in Figure 3.

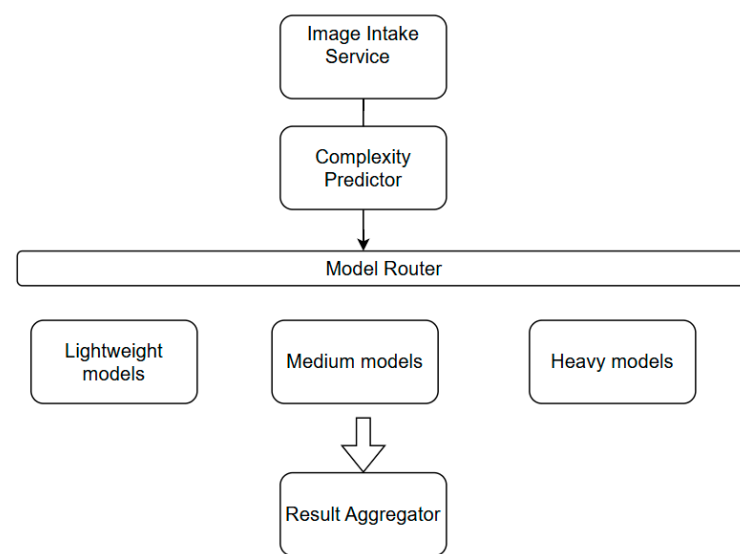


Figure 3. Pipeline architecture.

DataCollectionService is responsible for collecting, pre-processing, and managing image data used in all stages of the lifecycle. ModelTrainingService implements the process of training deep learning models for image processing tasks, such as classification, segmentation, and object detection. The service orchestrates the training of neural networks based on trained datasets. HyperparametersOptimizationService is responsible for automatically selecting hyperparameters of computer vision models to achieve the optimal balance between accuracy and computational complexity. MonitoringService provides continuous monitoring of computer vision models in production environments. It tracks performance metrics (latency, throughput), resource usage, and model quality. The ModelRegistryService is responsible for centrally storing, versioning, and managing computer vision models and their associated metadata. The service stores information about model architecture, training parameters, quality metrics, dataset versions, and deployment statuses.

Let us consider a dataset of the form:

$$D = \{(x_i, y_i)\}_{i=1}^N, \quad (1)$$

where x_i —image, y_i —corresponding class labels.

This formula takes into account three parameters to calculate the final complexity of the studied objects. One key factor is the number of objects in the image, because the primary goal of almost any classification task is to select objects and assign them to classes.

The criteria that determine the complexity of images include the following: The overall complexity of an image is determined by the formula

$$C(x_i) = T(x_i) + K(x_i) + S(x_i), \quad (2)$$

where $T(x_i)$ —dataset size;

$K(x_i)$ —number of objects in the image. The calculation of quantitative characteristics is performed using standard computer vision algorithms, in particular, lightweight segmentation methods, such as thresholding.

$S(x_i)$ —image size.

The formula calculates the total complexity of a dataset:

$$C_D = \frac{1}{N} \sum_{i=1}^N C(x_i) \quad (3)$$

Based on the value C_D , classification is introduced: low difficulty, medium difficulty, and great difficulty.

A variety of models are used for adaptive selection:

$$M = \{M_1, M_2, \dots, M_n\} \quad (4)$$

where M_i —neural network model

Considering the complexity category of the dataset, the optimal model is determined according to the rule

$$M^*(D) = \begin{cases} M_{light}, & C_D < \tau_1 \\ M_{mid}, & \tau_1 \leq C_D < \tau_2 \\ M_{heavy}, & C_D > \tau_2 \end{cases} \quad (5)$$

where τ_1 and τ_2 are threshold values that can be set manually depending on the type of task, based on the complexity of the task, the features of the objects under study, for example, micro-objects in biomedical images, or, in another case, cars on the road, in this case, we leave these parameters floating to provide greater flexibility for tuning.

The system functions as a self-configuring component within a microservice architecture, in which each stage (complexity analysis, accuracy prediction, model selection) is implemented by a separate service.

The proposed formal approach provides dynamic adaptation of the CNN architecture to the complexity of the input data and to the optimal ratio of performance and accuracy.

3.3. Ensemble Methods

The proposed pipeline also includes models for classifying quantitative features, not just images. The generalized classification algorithm using ensembles is as follows:

1. Loading a CSV file with prepared data divided into categories and classes;
2. Model pool definition via ModelProvider;
3. Adaptive BayesianModelSelector for selecting the best models;
4. Ensemble building with hard/soft voting;
5. Feature importance determination;
6. Graphical representation of results.

Therefore, the proposed algorithm is easily adaptable to any CSV file, since the parameters and classes are stored in the file itself without requiring additional settings. The module operates on the entire pool of models and automatically selects the optimal ensemble.

4. Results

4.1. Image Classification

The experiments were conducted using the Python (version 3.15) programming language and the machine learning libraries Keras (version 3) and TensorFlow (version 2.18), employing the CIFAR-10, CINIC-10, and IHCD BI [30] datasets. MobileNetV3-Small is a low-complexity model; ResNet-50 is medium-complexity and offers a balanced trade-off between accuracy and computational cost; while EfficientNet-B7 is high-complexity and achieves the highest accuracy.

The classification results and corresponding accuracy are given in Table 1.

Table 1. Classification results.

	CIFAR-10	CINIC-10	IHCDBI [30]
MobileNetV3	Accuracy: 0.9 Precision: 0.9 Recall: 0.9 F1: 0.9	Accuracy: 0.77 Precision: 0.78 Recall: 0.77 F1: 0.77	Accuracy: 0.84 Precision: 0.84 Recall: 0.84 F1: 0.84
ResNet-50	Accuracy: 0.88 Precision: 0.89 Recall: 0.88 F1: 0.88	Accuracy: 0.78 Precision: 0.78 Recall: 0.78 F1: 0.77	Accuracy: 0.87 Precision: 0.89 Recall: 0.87 F1: 0.85
EfficientNet-B7	Accuracy: 0.91 Precision: 0.91 Recall: 0.91 F1: 0.9	Accuracy: 0.79 Precision: 0.79 Recall: 0.79 F1: 0.79	Accuracy: 0.92 Precision: 0.92 Recall: 0.92 F1: 0.92

The following metrics were chosen to assess the classification quality: Accuracy, Precision, Recall, F1. For the CIFAR-10 dataset, the best results were shown by the MobileNetV3 and EfficientNet models. This suggests that using a lighter model gives almost the same results as a deep one. For the IHCD BI dataset, the EfficientNet-B7 model demonstrated significantly better results.

The proposed approach minimizes computational costs by dynamically selecting the least expensive model for a given dataset. The ROC curve and confusion matrix for the CIFAR-10 dataset based on the ResNet-50 model are shown in Figure 4.

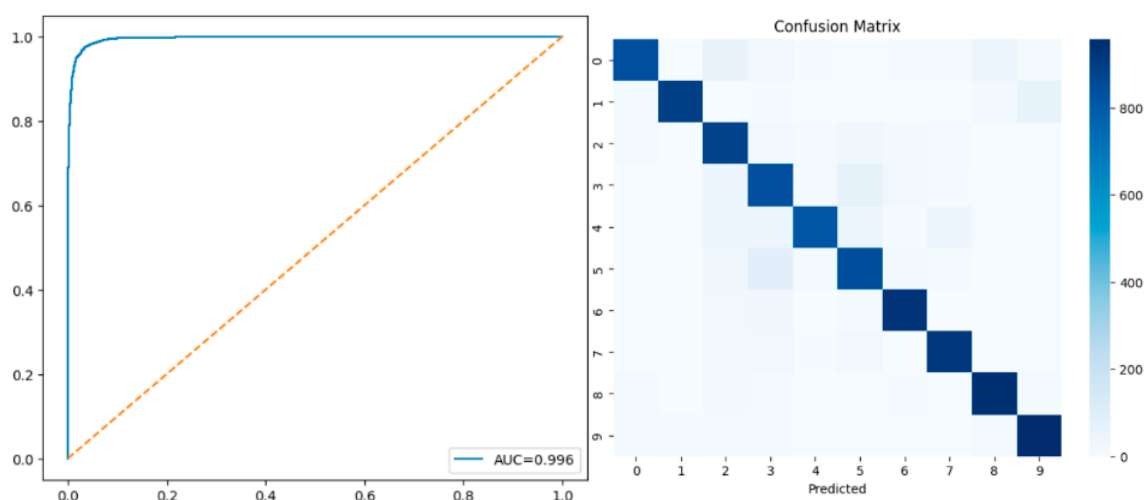


Figure 4. ROC curve and confusion matrix for the CIFAR-10 dataset based on the ResNet-50 model.

The AUC indicator is 0.99 on the CIFAR-10 dataset, confirming the model's feasibility. The confusion matrix confirms the above results.

The classification results using the ResNet-50 model based on the CINIC-10 dataset are shown in Figure 5.

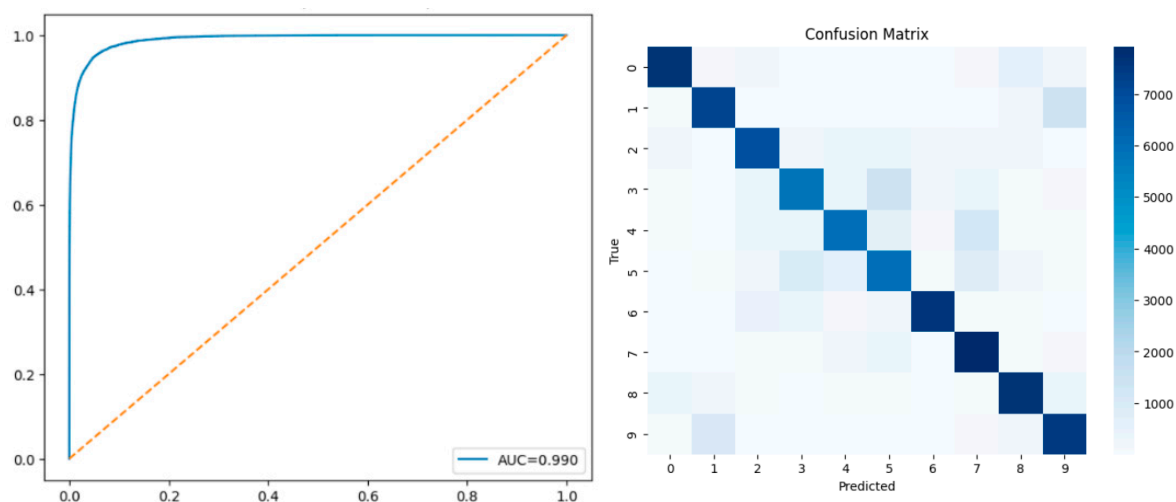


Figure 5. ROC curve and confusion matrix for the CINIC-10 dataset based on the ResNet-50 model.

Using the CINIC-10 dataset with the ResNet-50 model yields an AUC of 0.990.

Figure 6 shows the classification results for the four classes of the IHCDBI dataset [30,31].

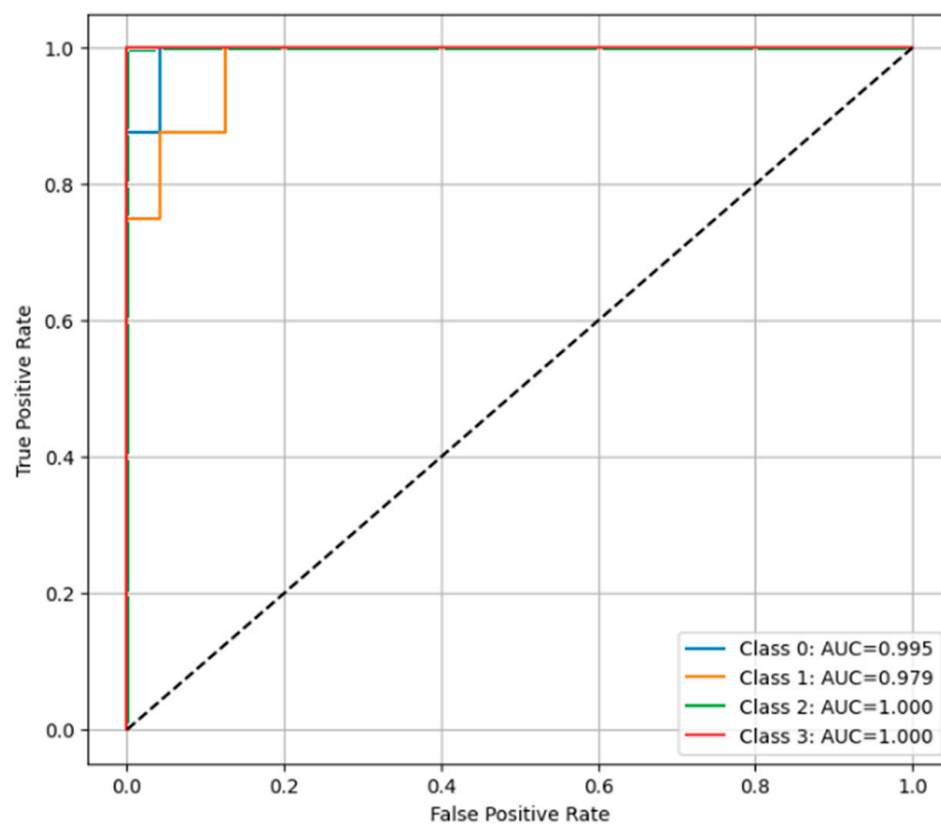


Figure 6. ROC curve for four classes of cytology dataset.

The results obtained in this article are presented to confirm the need to find the optimal model for different datasets and this demonstrates that depending on the type of data, it is possible to use lighter models and accordingly save resources. Based on

experiments conducted with different models, the complexity of the training process was analyzed across datasets of varying complexity. Dataset complexity was determined using three criteria that primarily influence execution time. The evaluated models were also categorized into three levels of complexity according to network depth.

In general, the results indicate that for relatively simple datasets, it is advisable to employ deep neural network architectures, as they enable high accuracy to be achieved within an acceptable training time.

4.2. Ensemble Classification

In this work, an adaptive selection mechanism for an ensemble-based classification algorithm is implemented. This allows you to choose the best result and obtain stable results. The module consists of the following algorithms: logistic Regression, SVM, Random Forest, XGBoost, KNN, Gradient Boosting, and Naive Bayes.

In the final case, Hard Voting and Soft Voting are used. Bayesian optimization is a method for automatically searching for optimal parameter values.

In the process of classification:

- Models are evaluated independently;
- The best models are selected;
- The selected models form an ensemble.

The Bayesian optimizer searches for the optimum more efficiently than random search.

The selected dataset comprises descriptions of microobjects in cytological images. The number of studied objects is 800 items.

The correlation matrix is shown in Figure 7.

	contour_area	contour_perimetr	contour_circularity	aspect_ratio	class
contour_area	1.000000	0.937447	1.000000	−0.159061	−0.204928
contour_perimetr	0.937447	1.000000	0.937447	−0.095697	−0.137371
contour_circularity	1.000000	0.937447	1.000000	−0.159061	−0.204928
aspect_ratio	−0.159061	−0.095697	−0.159061	1.000000	0.034885
class	−0.204928	−0.137371	−0.204928	0.034885	1.000000

Figure 7. Correlation matrix of cytological microobjects.

The colors show the direction and strength of the correlation between the variables. The more intense the red, the stronger the direct relationship. The darker the blue, the stronger the inverse relationship. The results of the correlation analysis showed the presence of strong multicollinearity between the geometric features contour_area, contour_perimeter, and contour_circularity, the correlation coefficients between which exceed 0.93, and an almost perfect linear relationship is observed between contour_area and contour_circularity.

Model Feature Importance is given in Table 2. Analyzing this table, we can conclude that, for processing quantitative characteristics of cytological images, the parameter contour_area is the most significant.

The geometric features contour_area and contour_circularity have the highest weight, together providing over 60% of the total importance. This indicates that the size and shape of objects are key factors for classification. The best results of the ensemble models are given in Table 3.

Table 2. Model Feature Importance.

Feature	Importance
contour_area	0.309368
contour_circularity	0.295311
contour_perimetr	0.260561
aspect_ratio	0.134759

Table 3. Model Feature Importance. Classification results.

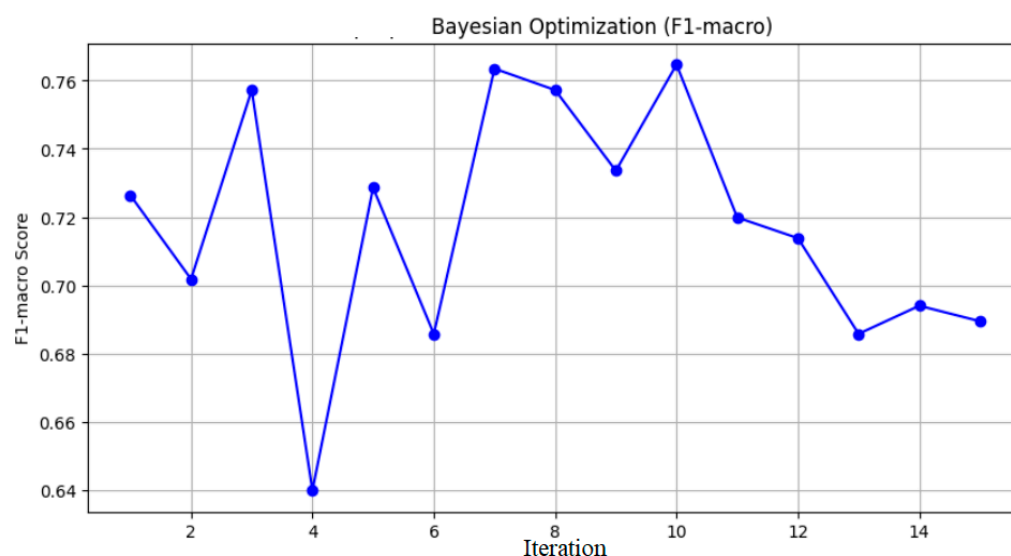
Algorithm	rf	gb	lr	svc
	0.94	0.98	0.6	0.49

The following final result was obtained for the voting ensemble from the models: ['rf', 'gb', 'lr']:

Accuracy: 0.7875

F1-macro: 0.7647

Results of using Bayesian Optimization (F1-macro) are shown in Figure 8.

**Figure 8.** Results of using Bayesian Optimization (F1-macro).

The graph shows the change in the value of the F1-macro metric during the Bayesian optimization of the model's hyperparameters. At the initial steps, significant variability in the F1-macro values is observed, which indicates an active exploration of the hyperparameter space and the search for promising areas. The results of Bayesian hyperparameter optimization demonstrated rapid convergence of the model, with the maximum F1-macro value being achieved in 10 iterations.

Figure 9 shows the graphical interface of the developed web part of the pipeline. The figure shows the graphical interface of the data classification software system. The interface is oriented towards interactive work with the user and provides experiment from selecting a data set to obtaining classification results.

This modular interface structure ensures ease of experimentation, reproducibility of results, and simplifies comparative analysis of different models.

A comparative analysis of the developed pipeline with analogues is given in Table 4.

Figure 9. Graphical interface of the developed web part of the pipeline.

Table 4. A comparative analysis of the developed pipeline with analogues.

Parameter	Kubeflow Pipelines	Apache Airflow	Proposed Pipeline
Type of architecture	Kubernetes-based	DAG-workflow	Microservice architecture with autonomous modules
Scalability	High	Medium	High, dynamic autoscalability
Service isolation	Partial	Limited	Complete isolation through lightweight services
Automation	High	High	Autonomous optimization
Flexibility of integrations	Kubernetes orientation	Mixed scenarios	Multi-environment, multi-cloud integration
Adaptability of ML processes	+/-	+/-	Automatic model and configuration selection
Support for model auto-tuning	third-party components	third-party components	Built-in AutoModelSelector component
Image orientation (CV)	-	-	Special task complexity profiles and CNN architectures

The proposed method achieves accuracy comparable to NAS, while requiring an order of magnitude fewer resources. In microservice systems, the approach provides optimal latency and throughput through adaptive model selection.

Unlike existing MLOps solutions that primarily focus on deployment and monitoring, the proposed approach introduces a fully adaptive ML pipeline that dynamically selects optimal deep learning models. Table 5 provides a comparative analysis of existing pipelines with the proposed one.

Table 5. Comparison of the proposed adaptive ML pipeline with existing approaches.

Feature	Static ML Pipeline	Traditional MLOps Solutions	Proposed Pipeline
Dynamic model selection	-	Limited	+
Data complexity awareness	-	-	+
Adaptive computation graph	-	-	+
			High
Support for high-load systems	Limited	Moderate	
Ensemble-based complexity classification	-	-	+

Classical deep learning pipelines [32] rely on static model architectures and lack adaptability to data complexity or system load. Traditional MLOps frameworks [33] primarily address deployment, monitoring, and reproducibility, but do not support dynamic model selection based on formal data characteristics. The proposed approach integrates data complexity estimation and resource-aware model selection within a unified ML pipeline.

The model selection procedure operates over a finite set of candidate models and has linear complexity $O(K)$, where K is the number of available architectures. The ensemble-based data complexity classification module introduces a negligible overhead compared to CNN inference. Complexity comparison of the proposed adaptive ML pipeline and benchmark approaches shown in Table 6.

Table 6. Complexity comparison of the proposed adaptive ML pipeline and benchmark approaches.

Approach	Model Selection Complexity	Pipeline Overhead	Scalability
Static CNN pipeline	$O(1)$	Low	Limited
Traditional MLOps	$O(1)$	Moderate	Moderate
Resource-aware inference	$O(k)$	Low	Moderate
			High
Proposed pipeline	$O(k)$	Moderate	

The results of deploying the project using Terraform (version 1.15) on the DigitalOcean cloud service are shown in Table 7. The process includes the time to create a droplet on the DigitalOcean cloud service, install all necessary libraries, and load the dataset, which takes the longest. The deployment process uses an IaC approach, with the main infrastructure code defined in a configuration file.

Table 7. Complexity comparison of the proposed adaptive ML pipeline and benchmark approaches. Comparative analysis of time to deploy from scratch.

Approach	Time to Deploy a Project from Scratch (Before the Launch Stage)
CNN	3–6 min
Unet	4–8 min
Ensemble methods (quantitative characteristics)	2–4 min

Existing adaptive MLOps solutions based on Kubeflow Pipelines, combined with Neural Architecture Search, primarily focus on automating model and architecture selection during the training phase. In such systems, adaptivity is achieved through offline exploration of an ample architecture search space, which requires repeated model training

and validation. As a result, these approaches incur substantial computational overhead and are typically executed on large-scale infrastructure. Static ML pipelines use a fixed deep learning model selected manually or empirically before deployment. The proposed adaptive ML pipeline differs fundamentally from both approaches by enabling online adaptivity at runtime. Instead of performing computationally expensive architecture search, the proposed system dynamically selects an appropriate model from a predefined pool based on ensemble-driven data complexity estimation and real-time resource monitoring.

The runtime in real mode is much less, but this table highlights the deployment process from scratch.

The experiments were conducted in a DigitalOcean cloud service, using the Ubuntu (version 22) operating system, the Python (version 3) programming language, and libraries for training neural network models.

The computational complexity of the proposed adaptive ML pipeline is evaluated using real-world CNN architectures, including the ResNet and EfficientNet families. In contrast, static pipelines rely on a fixed model with constant inference complexity.

5. Discussion

Automatic architecture search (NAS) methods build the optimal model by searching the entire architecture space or a subset of it. Searching for an NAS model often requires thousands of GPU hours, whereas the proposed approach does not perform an architecture search but instead selects a model from a predefined set. Since NAS requires an expensive search, it cannot work in real-time systems or in microservice architectures.

The proposed approach provides a formally described decision-making mechanism.

In multi-model cascades, the complexity is determined by a single input frame. The proposed approach estimates the complexity distribution of the entire dataset, enabling an optimal choice before training.

Lightweight approaches focus only on optimizing the weights of a single model, rather than on selecting the appropriate model from a set of different ones, and there is also no mechanism for adapting to the complexity of the data.

The proposed algorithm introduces a formal dataset-complexity metric as the basis for selecting a CNN, which is more efficient for highly loaded microservice systems than NAS. It guarantees a better balance between performance and accuracy compared to lightweight or cascade approaches.

The architecture [1] divides the system into independent microservices responsible for distinct functions, such as content, analytics, recommendations, and model management, thereby confirming the relevance of this task. The results presented in [4] demonstrate the relevance of using ML pipelines to implement flexibility in the design of software systems using machine learning.

The system for LLM processing proposed in [6] confirms the relevance of developing such pipelines, but does not fully describe the hardware component and storage for storing different types of data.

The importance of data preprocessing is discussed in [11]. Taking into account previous experience to ensure improved quality and the final result, this work separates the preprocessing module into a service that adapts input data, in particular images, to a standardized form.

The importance of using a microservice approach to designing big-data processing systems with deep learning elements is discussed in [12]. The work systematizes modern software development methods and shows how the combination of agile processes, cloud tools, DevOps, microservices, and AI forms a new, more efficient ecosystem for creating software products. In [13], the authors proposed an architecture with high-load elements,

but unlike this work, they did not propose a mechanism for selecting the type and model of a neural network to maximize performance.

In many cases, the use of a monolithic architecture is sufficient, and the main comparative characteristics are considered in [20]; however, as a result of the analysis, we found that for the tasks of processing big data in the form of media files using deep learning, a microservice architecture is better suited. Recently, the development of technologies that integrate the Internet of Things and artificial intelligence has been advancing rapidly. The paper [34] presents a federated edge intelligence paradigm that integrates edge computing with federated learning (FL) to ensure secure and efficient medical data processing. The relevance of developing software deployment pipelines for medical applications that incorporate deep learning is further examined in [13,35].

The proposed ML pipeline is implemented as a cloud-native microservice architecture, where each functional component is deployed as an independent service. Load balancing is performed at the API gateway or service mesh level, distributing incoming requests across multiple service instances to ensure stable performance under varying workload conditions. Fault recovery is ensured through container orchestration mechanisms that continuously monitor service health and automatically restart failed instances.

In conclusion, the analysis of existing studies highlights the importance of developing software solutions that leverage deep learning through pipeline-based architectures and mechanisms to simplify selecting machine learning models for image classification and quantitative feature analysis. The primary advantage of this approach lies in reducing the number of trial computations while generally improving accuracy.

6. Conclusions

The developed pipeline enables the selection of the most effective model based on the current system load and available computational resources. Support for a scalable multi-model inference infrastructure is significant for computationally demanding image segmentation and object detection tasks.

Applying this approach to image processing tasks, especially segmentation and recognition, enables high model accuracy in real time while efficiently processing large volumes of data.

In this paper, a self-optimizing mechanism is developed to select the optimal convolutional neural network model based on the dataset's input characteristics.

Author Contributions: Conceptualization, O.P. and M.S.; methodology, O.P.; software, O.P.; validation, O.P. and M.S.; formal analysis, O.P.; investigation, O.P. and M.S.; resources, O.P.; data curation, M.S.; writing—original draft preparation, O.P.; writing—review and editing, O.P.; visualization, M.S.; supervision, O.P.; project administration, O.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are openly available in Cytological and histological images of breast cancer at <https://doi.org/10.5281/zenodo.7890874>, reference number [30].

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DIS	Data Intake Service
APS	Adaptive Pre-processing Service
MSRS	Model Selection & Routing Service
NAS	Neural Architecture Search
API gateway	A central entry point that sits between clients and backend services
Grafana	Open source analytics & monitoring solution for every database
Prometheus	Collects and stores time-series metrics (like CPU, memory)
Mysql	Free relational database management system
Docker	Toolkit for managing isolated Linux containers

References

1. Artamonov, Y.; Plotytsia, S.; Radchenko, K.; Kotsiur, A. Microservice architecture of intelligent educational platforms with elements of ml pipeline self-optimization. *Sci. Technol.* **2025**, *10*, 1059–1073. [\[CrossRef\]](#)
2. Sharma, P.; Sarangdevot, S.S. Optimizing Machine Learning Pipelines via Adaptive Hybrid Classification Models: Toward Scalable, Self-Updating AI Architectures. *Int. J. Sci. Res. Sci. Eng. Technol.* **2025**, *12*, 84–96. [\[CrossRef\]](#)
3. Yin, H.; Jin, D.; Hong, H.; Moon, J.; Gu, Y.H. IAQ-STL-ML: A novel indoor air quality prediction pipeline using a meta-learning framework with STL decomposition. *Environ. Technol. Innov.* **2025**, *38*, 104107. [\[CrossRef\]](#)
4. Gómez, C.; López, L.; Ayala, C.; López, M. MLSToolbox Code Generator: A tool for generating quality ML pipelines for ML systems. *SoftwareX* **2025**, *32*, 102379. [\[CrossRef\]](#)
5. DeGroat, W.; Venkat, V.; Pierre-Louis, W.; Abdelhalim, H.; Ahmed, Z. Hygieia: AI/ML pipeline integrating healthcare and genomics data to investigate genes associated with targeted disorders and predict disease. *Softw. Impacts* **2023**, *16*, 100493. [\[CrossRef\]](#)
6. Khattab, O.; Singhvi, A.; Maheshwari, P.; Zhang, Z.; Santhanam, K.; Vardhamanan, S.; Haq, S.; Sharma, A.; Joshi, T.T.; Moazam, H.; et al. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *arXiv* **2023**, arXiv:2310.03714. [\[CrossRef\]](#)
7. Tamim, I.; Shami, A.; Ong, L. ML-Based Strategies to Optimize O-RAN VNFs for Latency and Reliability. In Proceedings of the 2023 IEEE Future Networks World Forum (FNWF), Baltimore, MD, USA, 13–15 November 2023; pp. 1–7.
8. Al Marouf, A.; Ahmed; Rokne, J.G.; Alhaji, R. Identification of Potential Biomarkers in Prostate Cancer Microarray Gene Expression Leveraging Explainable Machine Learning Classifiers. *Cancers* **2025**, *17*, 3853. [\[CrossRef\]](#)
9. Vincenzo, L.; Coelli, S.; Gorlini, C.; Forzanini, F.; Rinaldo, S.; Andreasi, N.G.; Romito, L.; Eleopra, R.; Bianchi, A.M. The Role of MER Processing Pipelines for STN Functional Identification During DBS Surgery: A Feature-Based Machine Learning Approach. *Bioengineering* **2025**, *12*, 1300. [\[CrossRef\]](#)
10. Conte; Luana; De Nunzio, G.; Raso, G.; Cascio, D. Multi-Class Segmentation and Classification of Intestinal Organoids: YOLO Stand-Alone vs. vs. Hybrid Machine Learning Pipelines. *Appl. Sci.* **2025**, *15*, 11311. [\[CrossRef\]](#)
11. Koukaras; Paraskevas; Tjortjis, C. Data Preprocessing and Feature Engineering for Data Mining: Techniques, Tools, and Best Practices. *AI* **2025**, *6*, 257. [\[CrossRef\]](#)
12. Thilagavathy, R.; Veeramani, T.; Sundaravadivazhagan, B.; Deebalakshmi, R. Evolution of software engineering: From traditional to modern approaches. In *Advances in Computers*; Elsevier: Amsterdam, The Netherlands, 2025. [\[CrossRef\]](#)
13. Pitsun, O.; Shymchuk, M. A high-load architecture for image processing based on microservices. In Proceedings of the CIAW-2025: Computational Intelligence Application Workshop, Lviv, Ukraine, 26–27 September 2025.
14. Zhu, J.; Bai, W.; Zhang, H.; Lin, W.; Zhou, T.; Li, K. Adaptive multi-objective swarm intelligence for containerized microservice deployment. *Future Gener. Comput. Syst.* **2026**, *174*, 108012. [\[CrossRef\]](#)
15. Ponce, F.; Verdecchia, R.; Miranda, B.; Soldani, J. Microservices testing: A systematic literature review. *Inf. Softw. Technol.* **2025**, *188*, 107870. [\[CrossRef\]](#)
16. Kaushik, N.; Kumar, H.; Raj, V. A systematic review of QoS enhancement techniques in microservices architecture. *Comput. Electr. Eng.* **2025**, *127*, 110550. [\[CrossRef\]](#)
17. Alshuqayran, N.; Ali, N.; Evans, R. A model-driven architecture approach for recovering microservice architectures: Defining and evaluating MiSAR. *Inf. Softw. Technol.* **2025**, *186*, 107808. [\[CrossRef\]](#)
18. Oyeniran, O.C.; Adewusi, A.O.; Adeleke, A.G.; Akwawa, L.A.; Azubuko, C.F. Microservices architecture in cloud-native applications: Design patterns and scalability. *Comput. Sci. IT Res. J.* **2024**, *5*, 2107–2124. [\[CrossRef\]](#)
19. Lucian, A.C.; Bocu, R. Authentication Challenges and Solutions in Microservice Architectures. *Appl. Sci.* **2025**, *15*, 12088. [\[CrossRef\]](#)

20. Al-Qora'n, L.F.; Ahmad, A.A.-S. Modular Monolith Architecture in Cloud Environments: A Systematic Literature Review. *Future Internet* **2025**, *17*, 496. [CrossRef]
21. Yuanbo, L.; Li, Y.; Wang, G.; Hu, H. An Adaptive Dynamic Defense Strategy for Microservices Based on Deep Reinforcement Learning. *Electronics* **2025**, *14*, 4096. [CrossRef]
22. Hossam, H.; Abdel-Fattah, M.A.; Mohamed, W. A Pattern-Based Framework for Automated Migration of Monolithic Applications to Microservices. *Big Data Cogn. Comput.* **2025**, *9*, 253. [CrossRef]
23. Rakshith, S.; Sierla, S.; Vyatkin, V. From DevOps to MLOps: Overview and Application to Electricity Market Forecasting. *Appl. Sci.* **2022**, *12*, 9851. [CrossRef]
24. CobParro; Carlos, A.; Lalangui, Y.; Lazcano, R. Fostering Agricultural Transformation through AI: An Open-Source AI Architecture Exploiting the MLOps Paradigm. *Agronomy* **2024**, *14*, 259. [CrossRef]
25. Andrej, R.; Kotuliak, I.; Sobolev, D. Evaluating Deployment of Deep Learning Model for Early Cyberthreat Detection in On-Premise Scenario Using Machine Learning Operations Framework. *Computers* **2025**, *14*, 506. [CrossRef]
26. Park, Y.; Mun, J.; Lee, Y.; Um, J.; Choi, J.; Choi, J. Data-Driven Optimization of Healthcare Recommender System Retraining Pipelines in MLOps with Wearable IoT Data. *Sensors* **2025**, *25*, 6369. [CrossRef]
27. Daniel, Z.R.; Bărbulescu, C.; Constantinescu, R. A Practical Approach to Defining a Framework for Developing an Agentic AIOps System. *Electronics* **2025**, *14*, 1775. [CrossRef]
28. Caruana, R.; Niculescu-Mizil, A.; Crew, G.; Ksikes, A. Ensemble selection from libraries of models. In Proceedings of the Twenty-First International Conference on Machine Learning (ICML '04), Banff, AL, Canada, 4–8 July 2004; Association for Computing Machinery: New York, NY, USA, 2004; p. 18. [CrossRef]
29. Yao, Y.; Pirš, G.; Vehtari, A.; Gelman, A. Bayesian Hierarchical Stacking: Some Models Are (Somewhere) Useful. *Bayesian Anal.* **2022**, *17*, 1043–1071. [CrossRef]
30. Berezsky, O.; Datsko, T.; Melnyk, G. *Cytological and Histological Images of Breast Cancer [Data Set]*; Zenodo: Geneva, Switzerland, 2023. [CrossRef]
31. Database. IHCDBI Digital Immunohistochemical Image Database of Breast Cancer. /10.05.2023 Bulletin No. 76. Dated 31 July 2023//Copyright Registration Certificate Number 118979. Available online: <https://iprop-ua.com/cr/0r6kml00/> (accessed on 4 January 2026).
32. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
33. Yang, C.; Wang, W.; Zhang, Y.; Zhang, Z.; Shen, L.; Li, Y.; See, J. MLife: A lite framework for machine learning lifecycle initialization. *Mach Learn.* **2021**, *110*, 2993–3013. [CrossRef]
34. Zhang, C.; Shan, G.; Roh, B.H.; Zhu, F.; Jiang, J. FEI-Hi: Federated Edge Intelligence for Healthcare Informatics. *IEEE J. Biomed. Health Inform.* **2025**. Preprint. [CrossRef]
35. Berezsky, O.; Pitsun, O.; Melnyk, G.; Derysh, B.; Liashchynskyi, P. Application Of MLOps Practices for Biomedical Image Classification. In Proceedings of the 5th International Conference on Informatics & Data-Driven Medicine, Ceur Workshop Proceedings, Bratislava, Slovakia, 17–19 November 2023; Volume 3302, pp. 69–77.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.