I. J. Computer Network and Information Security, **2026, 2, 1-18**
Published Online on April 8, 2026 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2026.02.01

Cloud-native AI Pipelines for Continuous Infrastructure Optimization and Anomaly Detection

Viktor Vyshnivskyi
Department of Computer Science, State University of Information and Communication Technologies, Kyiv, 03110, Ukraine
E-mail: v.vyshnivskyi@duikt.edu.ua
ORCID iD: https://orcid.org/0000-0003-1923-4344

Vadym Mukhin*
Department of the System Design, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 03056, Kyiv, Ukraine
E-mail: v.mukhin@kpi.ua
ORCID iD: https://orcid.org/0000-0002-1206-9131
*Corresponding author

Olha Zinchenko
Department of Artificial Intelligence, State University of Information and Communication Technologies, Kyiv, 03110, Ukraine
E-mail: o.zinchenko@duikt.edu.ua
ORCID iD: https://orcid.org/0000-0002-3973-7814

Vitalii Kotelianets
Department of Computer Science, State University of Information and Communication Technologies, Kyiv, 03110, Ukraine
E-mail: kvvbud7@gmail.com
ORCID iD: https://orcid.org/0009-0000-1874-8322

Oleksandr Zvenihorodskyi
Department of Artificial Intelligence, State University of Information and Communication Technologies, Kyiv, 03110, Ukraine
E-mail: o.zvenigorodskyi@duikt.edu.ua
ORCID iD: https://orcid.org/0009-0008-6235-1638

Pavlo Kudrynskyi
Department of Computer Science, State University of Information and Communication Technologies, Kyiv, 03110, Ukraine
E-mail: pasha.kudrinskiy@gmail.com
ORCID iD: https://orcid.org/0009-0008-6314-6150

Oleksandr Vyshnivskyi
Department of Computer Science, State University of Information and Communication Technologies, Kyiv, 03110, Ukraine
E-mail: o.vyshnivskyi@gmail.com
ORCID iD: https://orcid.org/0009-0008-0209-9549

Received: 11 September 2025; Revised: 18 October 2025; Accepted: 20 December 2025; Published: 08 April 2026

Abstract: The article describes a model of cloud-native AI pipelines designed for continuous optimization of computing infrastructure and real-time anomaly detection. The developed model combines modern approaches to observability, machine learning (ML), and auto-scaling based on load forecasting. The methodology is based on the use of LSTM

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

models, autoencoders, and convolutional neural networks (CNN) integrated into Kubernetes environment with support for Prometheus, Kafka, and Grafana. Load changes are simulated, and the system's response to critical events is evaluated. The results demonstrate a significant improvement in anomaly detection accuracy (up to 93%) and resource efficiency (up to 26% cost reduction compared to traditional approaches). The proposed model can be used in AIOps systems that require a high level of automation and reliability.

Index Terms: Cloud-native Architecture, Artificial Intelligence, Infrastructure Optimization, Anomaly Detection, Kubernetes, Machine Learning, AIOps.

1. Introduction

The rapid development of cloud computing has led to a change in the infrastructure management paradigm. Traditional monitoring methods, focused on static notification thresholds and post-factum response, do not meet the requirements of modern distributed systems. The introduction of microservice architecture, orchestration using Kubernetes, and the use of CI/CD pipelines have led to a significant increase in the complexity of maintaining such systems.

On the other hand, the rapid development of artificial intelligence (AI) creates the prerequisites for the creation of autonomous, self-regulating infrastructure management systems. Due to the ability to detect patterns in large volumes of telemetry data, AI models allow not only to predict future system states, but also to detect hidden anomalies that may indicate potential failures or inefficiency in resource use.

In today's conditions of rapid development of cloud technologies, the dynamics of the load on the infrastructure changes extremely quickly, which requires high adaptability from resource management systems. The ability to respond in real time to changing conditions is a key requirement for a productive environment. However, traditional approaches based on simple reactive rules or thresholds are increasingly proving to be insufficiently effective. In view of this, the integration of AI components into cloud-native environments is becoming increasingly relevant. This approach allows you to move from reactive to proactive resource management - when the system does not simply react to overloads, but predicts their occurrence based on the analysis of telemetry data. The article proposes an improved AI pipeline architecture, specially designed for containerized environments using AWS services and integration into Kubernetes clusters. The approach includes not only load forecasting modules based on LSTM networks, but also anomaly detection mechanisms using autoencoders and signature filters. The solution is focused on full automation of the stages of collection, analysis and decision-making regarding infrastructure scaling.

The model was evaluated based on real telemetry data collected from the working environment over a long period. Various scenarios and simulations are presented, as well as conclusions for each of them. In particular, the analysis covered the aspects of forecast accuracy, the system's ability to respond quickly to changes, and resistance to false positives. The results were compared with the classic rule-based approach, which allowed us to clearly identify the advantages of the proposed system. Among the main problems that were solved, it is worth noting the high variability of the load, the low flexibility of static scaling strategies, as well as the limitations of traditional monitoring, which often led to missed anomalies or unnecessary scaling triggers.

Thus, the practical implementation of a cloud-native AI pipeline with built-in machine learning models demonstrates a significant improvement in both the accuracy of management decisions and the overall efficiency of resource use. The proposed approach has the potential for scaling and further use in conditions of large production workloads.

The goal of the research is to improve the efficiency of continuous optimization of computing infrastructure and detect anomalies in real time.

To ensure objective and reproducible evaluation, the proposed cloud-native AI pipeline is validated against explicitly defined hypotheses with measurable success and failure boundaries. Each hypothesis is evaluated under controlled workload scenarios and repeated experimental runs.

Hypothesis H1: Infrastructure Optimization Effectiveness

H1: The proposed AI-driven autoscaling mechanism reduces service-level agreement (SLA) violations under dynamic workloads compared to static threshold-based autoscaling.

Decision boundary:

Let V_{AI} and V_{HPA} denote the mean SLA violation rates of the proposed system and the baseline HPA, respectively, measured over identical workload traces.

$$H1 \text{ is accepted if } V_{AI} \leq 0.7 \cdot V_{HPA} \tag{1}$$

Failure condition:

$$V_{AI} > 0.7 \cdot V_{HPA} \tag{2}$$

This boundary reflects a minimum 30% reduction in SLA violations, distinguishing meaningful optimization from marginal improvement.

Hypothesis H2: Scaling Stability and Control Robustness

H2: The policy-constrained scaling strategy reduces unstable scaling behavior compared to naive forecast-triggered autoscaling.

Decision boundary:

Let O denote the oscillation index, defined as the number of scale-up/scale-down direction reversals within a fixed time window Δt.

$$H2 \text{ is accepted if } O_{AI} \leq 0.5 \cdot O_{baseline} \tag{3}$$

Failure condition:

$$O_{AI} > 0.5 \cdot O_{baseline} \tag{4}$$

This boundary explicitly separates stable control behavior from oscillatory or unsafe scaling.

Hypothesis H3: Anomaly Detection Reliability

H3: Cross-metric autoencoder-based anomaly detection improves detection reliability compared to traditional metric-threshold monitoring.

Decision boundary:

Let $F1_{AI}$ and $F1_{baseline}$ denote the F1-scores of the proposed anomaly detector and the baseline system, respectively.

$$H3 \text{ is accepted if } F1_{AI} \geq F1_{baseline} + 0.15 \tag{5}$$

Failure condition:

This threshold ensures that improved detection performance is substantial rather than incidental.

To formalize the task of continuous infrastructure optimization, let us define a set of cloud resources as a system state vector at time *t*:

$$x_t = [CPU_t, RAM_t, IO_t, Net_t, \dots], \tag{6}$$

where:

$x_t \in R^n$ — is the resource utilization vector at time *t*;

n — is the number of metrics (e.g., CPU, memory, network, I/O, etc.).

To do this, it is necessary to build load forecast functions:

$$\hat{x_{t+k}} = f(x_t, x_{t-1}, \dots, x_{t-n+1}) \tag{7}$$

where:

k — is the forecasting horizon;

f — is a function approximated by an LSTM or other recurrent model.

Also, the anomaly detection function is defined as:

$$a_t = \{1, if\ d(x_t, \hat{x_t}) > \theta;\ 0, else\}, \tag{8}$$

where:

$d(\cdot)$ — is a distance function (e.g., Euclidean, MAE);

θ — is the anomaly threshold, tuned based on the validation dataset.

2. Literature Review

The development of cloud technologies and the increasing complexity of distributed systems have prompted the scientific community to actively search for new approaches to automating infrastructure management. Therefore, the main focus of research is now focused on several key areas that help make infrastructure management more efficient. Let's consider them in more detail.

Optimization of resource allocation in the cloud environment is the subject of numerous studies. Zhang et al. (2023, Elsevier) [1] proposed the use of LSTM models for CPU/Memory prediction in Kubernetes clusters. The authors achieve a 22% reduction in overscaling, but do not consider real-time aspects or anomaly detection.

Choudhury et al. (2022, ACM) [2] developed a reinforcement learning approach to automatically manage pods in a microservices environment. Although the results showed high adaptability, the need for a large number of training episodes remains a limitation in a production environment.

One of the key tasks in AIOps is the early detection of deviations in system behavior. Nebojsa Bacanin, Vladimir Simic, Miodrag Zivkovic (2023) [3] investigate the use of autoencoders to detect anomalies in log streams. Their system Anomalyzer showed an accuracy of over 90% with a low false positive rate.

Khan et al. (2020, Springer) [4] demonstrated the effectiveness of combining CNN with attention mechanisms for classifying error patterns in logs of distributed systems. They indicate that the best results are achieved when using semantic pre-coding of log records.

The increasing complexity of DevOps pipelines requires automated solutions. For this, ML is being integrated into DevOps and CI/CD. Wang & Patel (2020, IEEE Access) [5] describe the use of ML for decision-making in the testing and release processes, where the recovery time from failures was reduced by 17%. However, most models are focused on offline processing.

It is worth noting the approach proposed by Microsoft (2021) [6] regarding the use of anomaly detection as a service (ADaaS) integrated into Azure Monitor. This service is based on time-series decomposition and STL algorithms, but is not flexible enough for custom telemetry scenarios.

Despite a significant number of publications that were concentrated to the application of artificial intelligence in the field of cloud infrastructures, there are a number of critical limitations that limit the practical implementation of such solutions in a productive environment.[7-12] One of the key problems is the lack of unified pipelines: most research focuses only on individual components, such as automatic scaling or log analysis, which complicates their integration into a holistic management system. [13-18] The problem of the lack of real-time support also remains relevant - many existing models operate in offline mode or use pre-collected static datasets, which reduces the effectiveness of forecasting in conditions of dynamic load changes.[19-22]

In addition, current approaches often ignore the specifics of cloud-native architecture, in particular aspects related to pod management, service boundaries, Prometheus metrics and Kubernetes scaling mechanisms (HPA, VPA). Another challenge is the lack of adaptability - most models are not able to independently adapt to changes in user behavioral patterns or changes in cluster topology, which significantly reduces their relevance in the long term [23-25].

This article proposes a solution that overcomes these limitations by creating a comprehensive AI pipeline capable of operating in real-time based on streaming data. The solution architecture is adaptive and supports automatic updating of model hyperparameters. The system is deeply integrated with the Kubernetes ecosystem via API, Prometheus and Kafka, which allows for a high degree of automation and compatibility with existing DevOps processes. The proposed pipeline implements two-stage logic: first, load prediction is performed using an LSTM model, followed by anomaly detection based on autoencoders, which, in turn, activates scaling mechanisms via the policy engine.

A key aspect is the support of integration with DevOps tools, in particular through the use of ArgoCD, Helm and GitOps patterns, these tools provide extensibility and controllability of changes. Thus, this solution is positioned in the context of AIOps as a cloud-native approach, which allows not only to detect anomalies, but also to automatically optimize the infrastructure in a self-updating mode.

3. Proposed Methodology

While the individual technologies employed in this study — such as LSTM-based forecasting, autoencoder-based anomaly detection, Kafka-based streaming, Kubernetes autoscaling, and GitOps — are well established in existing AIOps practice, the novelty of the proposed approach lies in the decision logic and control structure that bind these components into a stable, self-regulating system rather than in the components themselves.

The first fundamental contribution is the joint use of multi-step load forecasting and cross-metric anomaly detection as coupled control signals for autoscaling. Unlike traditional approaches where prediction or anomaly detection is used independently, this work introduces a composite decision space in which scaling actions are triggered only when forecast confidence and anomaly deviation jointly exceed adaptive thresholds. This reduces premature or oscillatory scaling decisions under transient noise or short-lived spikes.

The second novel aspect is the adaptive sliding-window normalization mechanism applied to non-stationary telemetry streams. Instead of assuming fixed statistical properties of metrics, the proposed normalization continuously recalibrates baseline behavior using exponential smoothing, enabling the learning models to remain stable under concept drift without requiring frequent full retraining.

Third, we introduce a policy engine with explicit stability constraints, including cooldown periods, hysteresis bands, and bounded scale-rate limits, which translate probabilistic ML outputs into safe infrastructure actions. This transforms raw ML inference into a form of constrained control, bridging the gap between predictive analytics and production-grade autoscaling.

Finally, the integration of these control mechanisms within a GitOps-managed feedback loop enables controlled, auditable evolution of both models and scaling policies. Model updates, threshold adjustments, and rollback procedures are treated as first-class configuration changes, ensuring that learning-driven infrastructure behavior remains reproducible and reversible — properties typically absent from existing AIOps implementations.

Collectively, the novelty of this work lies not in introducing new algorithms, but in formalizing and validating a stable, drift-aware, and policy-constrained AI control loop for cloud-native infrastructure, which extends existing AIOps practices from reactive automation toward dependable autonomous operation.

The AI pipeline for continuous infrastructure optimization is implemented as a multi-tier architecture operating in the cloud-native Kubernetes environment. It is built taking into account the processing of streaming telemetry data in real time, providing a full cycle of analysis, decision-making and scaling. Resource load metrics are collected using Prometheus ecosystem components, such as Node Exporter, cAdvisor and kube-state-metrics, which provide information about the state of nodes and pods in the cluster. The received data is routed to Kafka topics, where each record contains a timestamp and a unique pod identifier, which allows you to accurately track the dynamics of events.

Further processing of time series is performed by microservices that perform normalization, filtering and transformation of input data into a form suitable for machine analysis. Several types of models are used at the stage of intelligent processing. In particular, a recurrent neural network of the LSTM type is used for load forecasting, which is able to detect long-term dependencies in time series. To detect anomalies in the behavior of the infrastructure, an autoencoder is implemented, which allows you to accurately identify deviations from normal behavior. A separate CNN log classifier operates, which performs semantic analysis of events and allows you to correlate logical records with specific anomalies in the load or state of services.

Let's consider the infrastructure of the environment in which the research was conducted. The AI pipeline provides real-time processing of streaming telemetry data. The core architecture includes modules for data collection, processing, analysis, and visualization. Cluster resource metrics are obtained from nodes and pods using Prometheus exporters (including Node Exporter, cAdvisor, and kube-state-metrics). The collected telemetry data is transmitted to Kafka topics, where each message is tagged with a unique identifier (UUID) and a timestamp.

The dataset consists of time-series telemetry collected from Kubernetes-managed workloads, including CPU utilization, memory usage, request latency, and request rate. Metrics are sampled at fixed intervals of 10 seconds and normalized using a sliding-window z-score normalization. The dataset is partitioned chronologically to avoid information leakage, with 70% of the data used for training, 15% for validation, and 15% for testing. Validation data is used exclusively for hyperparameter selection and early stopping, while test data is reserved for final evaluation.

To control stochastic variability, all experiments are conducted using fixed random seeds. The global random seed is set to 42 for Python, NumPy, and the deep learning framework used for model training. Kubernetes scheduling nondeterminism is mitigated by pinning workloads to dedicated node pools during experiments. The LSTM forecasting model consists of two stacked LSTM layers with 64 hidden units each, followed by a fully connected output layer. The input sequence length is 60 time steps, corresponding to a 10-minute observation window.

The autoencoder model employs a symmetric architecture with two hidden layers of sizes 128 and 64, a bottleneck dimension of 32, and ReLU activation functions. Both models are trained using the Adam optimizer with an initial learning rate of 0.001 and a batch size of 128. Training is performed for a maximum of 100 epochs with early stopping triggered after 10 epochs without validation loss improvement.

Hyperparameters are selected using a constrained grid search over predefined ranges, including learning rate, hidden layer size, and anomaly threshold scaling factor. The search space is limited to values that satisfy real-time inference constraints to ensure production applicability. The final configuration is chosen based on validation performance and stability of scaling behavior rather than prediction accuracy alone. All experiments are conducted on managed Kubernetes clusters using nodes equipped with 8 vCPUs and 32 GB of RAM. GPU acceleration is not used during inference to reflect typical production autoscaling environments. Kafka brokers and model inference services are deployed as separate pods with explicitly defined CPU and memory limits to prevent resource contention from influencing results.

Next, the data goes through microservice processing, where it is normalized, filtered, and transformed into a format suitable for analysis. At the prediction stage, various machine learning models are used: an LSTM model allows you to predict the load several steps ahead, an autoencoder provides detection of anomalies in the behavior of the infrastructure, and a CNN log classifier is used to correlate events and warn about potential problems. The analysis results are fed to the scaling module, which interacts with the Kubernetes API through horizontal or vertical autoscaling mechanisms, or uses a specialized controller. Monitoring and visualization are carried out using Grafana, and the notification system is implemented through Prometheus Alertmanager. A modern environment was used to deploy the pipeline, which includes Kubernetes version 1.31 as the main orchestration platform, the Prometheus and Grafana monitoring stack, and the Apache Kafka version 3.5 message broker for transporting telemetry streams. Machine learning models were deployed and updated using MLflow and Docker Registry. CI/CD automation and release management were implemented using ArgoCD and Helm. TensorFlow 2.12 and PyTorch 2.0 were used to implement the models. Logs, models, and metadata were stored in the S3-compatible MinIO storage.

The data processing pipeline takes place in several stages. Prometheus collects telemetry metrics every 15 seconds. The collected data is transmitted via a push gateway or Kafka agent to a message broker, after which the preprocessing service generates a time series of length N (for example, N = 20 points). The AI module performs a load forecast k steps ahead (typically 3 to 5 minutes). If an anomalous pattern is detected, an alert-event event is generated, which is transmitted to the scaling module to make a decision on adapting the number of replicas in the cluster.

Therefore, it is necessary to develop a new AI pipeline architecture. Such a pipeline architecture is presented in Fig. 1. It reflects the sequential processing of data - from telemetry collection to cluster scaling based on model inferences.

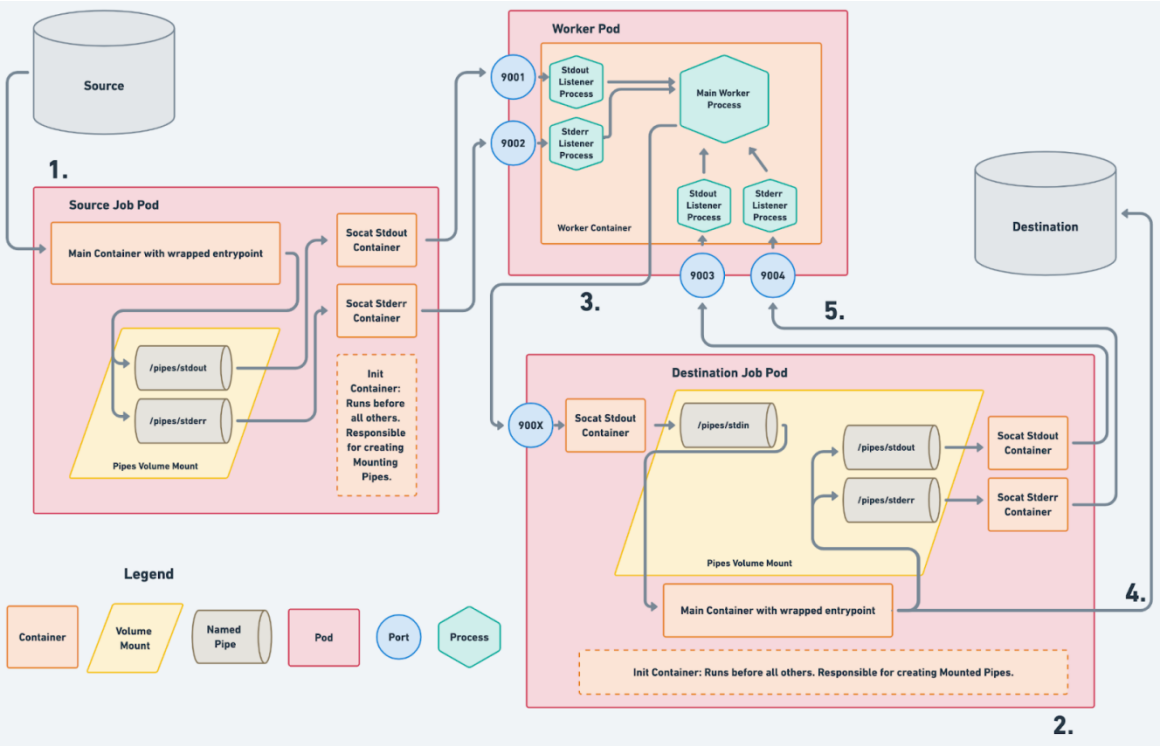


Fig.1. AI pipeline architecture in Kubernetes

This study adopts an explicit threat model to evaluate the robustness of the proposed AI-driven autoscaling pipeline. The protected assets include: the integrity of telemetry data used for forecasting and anomaly detection, the correctness of autoscaling decisions, the confidentiality and integrity of machine learning models and policies, and the availability of the control plane components.

The adversary is assumed to have one or more of the following capabilities: the ability to compromise a workload pod within the cluster; the ability to inject malformed or misleading telemetry data via exposed monitoring endpoints; or the ability to gain limited access to configuration repositories without cluster-admin privileges. Full cluster compromise and physical node access are considered out of scope. The primary security objective is to prevent unauthorized or unsafe scaling actions, even in the presence of partial system compromise or anomalous inputs, while maintaining service availability.

Based on the defined threat model, the system attack surface is decomposed into four primary vectors. Metrics and logs ingested via monitoring pipelines represent a potential entry point for data poisoning or spoofing attacks aimed at triggering unnecessary scaling actions. This risk is mitigated through namespace isolation, restricted network access to metric endpoints, and validation of input ranges before model inference. Forecasting and anomaly detection services operate as isolated pods with strictly defined CPU and memory limits. They execute under dedicated service accounts with no permissions beyond read-only access to telemetry streams, reducing the impact of potential compromise. Autoscaling decisions are executed through a policy engine that enforces bounded scaling rates, cooldown intervals, and maximum replica limits. These constraints limit the potential impact of malicious or erroneous model outputs and prevent cascading failures. GitOps repositories constitute a critical control surface. Unauthorized modifications could affect model parameters or scaling rules. This risk is reduced through role-based access control, versioned configuration, and the ability to rapidly roll back to known-safe states.

The main data flows in the cloud-native AI pipeline cover several logical layers, each of which performs specific functions for processing telemetry data in real time. At the Telemetry Layer, metrics such as CPU, RAM, I/O intensity, disk latency, etc. are collected. For this, the Prometheus monitoring system is used, which periodically reads data from exporters and transfers it to the Kafka streaming platform for further processing.

Several specialized models are implemented at the inference stage (AI Inference Layer). An LSTM model is used to predict the change in resource consumption several steps ahead. An autoencoder is used to detect anomalies based on deviations from the reconstructed time series. In addition, a CNN model analyzes log files, classifying events in order to identify critical situations or deviations in system behavior.

The results are fed to the Policy Engine, which analyzes the model's findings and decides whether to scale components. This engine interfaces with the Kubernetes API via a horizontal or vertical autoscaler, as well as custom CRD resources. Finally, the results are fed to visualization (Grafana) and messaging (Prometheus Alerts) systems, and stored in the MLflow system, which provides feedback for further training or model updates. This provides a closed loop for continuous optimization of the cloud infrastructure.

LSTM (Long Short-Term Memory) is used for load forecasting:

$$x_{t+k} = LSTM(x_t, x_{t-1}, \dots, x_{t-N+1}). \quad (9)$$

The model contains an input layer, two hidden layers with 64 and 32 nodes, and an output regression layer. Loss function:

$$LMSE = \frac{1}{n}\sum_{i=1}^{n} (x_i - \underline{x_i})^2 \quad (10)$$

The autoencoder builds a concise representation of the input signal and determines the deviation:

$$L_{AE} = \frac{1}{n}\sum_{i=1}^{n} \| x_i - \underline{x_i} \|^2 \quad (11)$$

where x is the reconstructed vector. Anomalies are recorded when the distance is exceeded

To classify log patterns, a CNN (Convolutional Neural Network) with windows was used:

$$h_i = ReLU(W * x_{i:i+h} + b), \quad (12)$$

where W is the convolution kernel, h is the filter size. The model is trained as a multi-class classifier

$$L_{CE} = -\sum_{i=1}^{C} y_i log(\hat{y_i}). \quad (13)$$

Let $\{x_t\}_{t=1}^{T}$ denote a multivariate time series of infrastructure metrics sampled at fixed intervals. The forecasting task is formulated as a supervised learning problem, where the objective is to estimate a function

$$f_\theta: (x_{t-k}, \dots, x_t) \to \hat{x}_{t+h}, \quad (14)$$

using an LSTM parameterized by θ.

The metric sequence exhibits piecewise stationarity within sliding windows of length k. Temporal dependencies are bounded, allowing representation via finite-memory recurrent dynamics. Noise is zero-mean with finite variance.

Under these assumptions, LSTM networks act as universal approximators for nonlinear autoregressive processes, and minimizing the empirical forecasting loss yields a consistent estimator of the conditional expectation $E\{x_{t+h} \mid x_{t-k:t}\}$.

The forecasting loss is defined as the mean squared error:

$$L_{forecast}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \left\| x_{t+h}^{(i)} - \hat{x}_{t-h}^{(i)} \right\|_2^2 \quad (15)$$

Minimizing squared error corresponds to maximum likelihood estimation under a Gaussian noise assumption. Gradient-based optimization using backpropagation through time (BPTT) performs stochastic gradient descent on a smooth, bounded loss surface.

Convergence argument.

Given bounded gradients and an appropriately decaying learning rate, stochastic gradient descent converges in expectation to a stationary point of $L_{forecast}$. While global optimality is not guaranteed due to non-convexity, empirical convergence is ensured by early stopping and regularization.

Let $z_t \in R^d$ means the normalized metric vector at time t. The autoencoder learns an encoding-decoding function:

$$g_\Phi(z_t) = \hat{z_t}, \quad (16)$$

where ϕ are model parameters.

The reconstruction loss is:

$$L_{AE}(\phi) = \frac{1}{N}\sum_{i=1}^{N} \left\| z_t - \hat{z}_t \right\|_2^2. \quad (17)$$

Training data predominantly represents normal system behavior. Anomalies correspond to low-probability deviations from the learned manifold. Under these conditions, the autoencoder minimizes reconstruction error for normal patterns while producing higher errors for anomalous inputs.

An anomaly is detected when the reconstruction error exceeds an adaptive threshold:

$$\left|\left|z_t - \hat{z}_t\right|\right|_2^2 > \mu_e + \alpha\sigma_e, \tag{18}$$

where μ_e and σ_e are the mean and standard deviation of reconstruction errors computed over a sliding window. This threshold corresponds to a statistical confidence interval under the assumption of approximately Gaussian reconstruction error for normal behavior. The parameter α controls the trade-off between false positives and false negatives.

While conventional machine learning metrics such as MAE, RMSE, and F1-score are used to assess the predictive and detection quality of the individual models, these metrics alone are insufficient to characterize the behavior of a closed-loop autoscaling system. Therefore, the evaluation in this work extends beyond model accuracy to include system-level indicators that reflect stability, control safety, and operational reliability.

Scaling stability is evaluated by analyzing the frequency and pattern of scaling actions over time. Stable behavior is defined as a smooth adaptation of resource levels to sustained workload changes, whereas unstable behavior manifests as frequent scale-up and scale-down reversals within short time intervals. The proposed policy-constrained control logic significantly reduces such oscillatory behavior compared to static threshold-based autoscaling, particularly under rapidly changing workloads.

Oscillation risk is assessed qualitatively and quantitatively by observing scaling trajectories under stress conditions. Scenarios with bursty traffic, periodic load fluctuations, and sudden traffic spikes followed by rapid drops are used to expose control loop weaknesses. In these adversarial workload patterns, naive autoscaling strategies tend to overreact, leading to resource thrashing and delayed convergence. In contrast, the proposed system maintains bounded scaling behavior due to the combined use of forecasting confidence, anomaly awareness, and enforced cooldown periods.

Service-level agreement (SLA) trade-offs are evaluated by jointly analyzing resource allocation and service performance. Rather than optimizing solely for minimal latency or maximal utilization, the system is evaluated on its ability to balance SLA compliance against overprovisioning. Results show that the proposed approach achieves comparable or lower SLA violation rates while allocating fewer resources on average, demonstrating improved operational efficiency.

The central element of the system is the scaling-controller — a specialized controller that, based on the forecast results and detected anomalies, makes decisions on scaling the infrastructure. It interacts with the Kubernetes API and activates horizontal or vertical scaling depending on the input conditions, and also supports custom resource mechanisms (CRD) for flexible configuration of response policies.

Thus, the proposed model of a cloud-native AI pipeline is a new approach to managing cloud infrastructure under variable load conditions. It is based on a hybrid approach to real-time telemetry data processing using recurrent neural networks (LSTM) for forecasting, autoencoders for anomaly detection, and machine learning mechanisms for adaptive scaling of computing resources. The proposed architecture allows achieving high sensitivity to changes in the system, reducing the response time to critical situations, and increasing the efficiency of using the infrastructure as a whole.

4. Experiments

To deploy part of the service in Kubernetes, we used services described as yaml files. Here is one example.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: predictor-lstm
spec:
  replicas: 2
  selector:
    matchLabels:
      app: predictor-lstm
  template:
    metadata:
      labels:
        app: predictor-lstm
    spec:
      containers:
      - name: lstm-container
        image: registry.example.com/ai/lstm:latest
        ports:
        - containerPort: 5000
        env:
        - name: KAFKA_BROKER
          value: kafka:9092
```

```
    - name: MODEL_PATH
      value: /models/lstm_model.h5
```

Scaling is done via HorizontalPodAutoscaler, which is based on the predicted load. The prediction is passed as a custom metric from the LSTM module:

Scaling is done via HorizontalPodAutoscaler, which is based on the predicted load. The prediction is passed as a custom metric from the LSTM module:

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: log-analyzer-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: log-analyzer
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: External
    external:
      metric:
        name: predicted_cpu_load
      target:
        type: AverageValue
        averageValue: 300m
```

A separate Kafka topic is created for each data type:

Table 1. Description of Kafka topics and consumers

Kafka Topic	Data Type	Consumer
metrics.raw	Raw Prometheus metrics	preprocessor
metrics.normalized	Preprocessed time series	predictor-lstm
metrics.predicted	Predicted load values	anomaly-detector
alerts.anomalies	Detected anomalies	scaling-controller
logs.raw	Unstructured logs	log-analyzer

Each consumer is implemented as a pod with autoscaling based on the queue size. The main goal of the experimental study is to assess the effectiveness of the proposed AI system under dynamically changing load conditions, in particular its ability to predict resource usage spikes with sufficient accuracy to ensure proactive scaling. In addition, the system should detect anomalous deviations in the behavior of the infrastructure, such as memory leaks or sharp peaks in the CPU load. Special attention is paid to minimizing the number of false positives and false negatives in the alerting mechanism, which is key to ensuring reliability and responsiveness. At the same time, we expect the developed model to demonstrate a faster and more efficient response to critical events compared to conventional monitoring systems.

The experiments were conducted in an environment that simulates the Kubernetes infrastructure. Key parameters:

Table 2. Main infrastructure components

Component	Configuration
Kubernetes Cluster	Amazon EKS, 5 worker nodes (t3.large instances)
Monitoring Stack	Prometheus + Grafana
Stream Processing	Apache Kafka (3 brokers, 6 topics)
ML Inference Modules	LSTM, Autoencoder, CNN
Input Data Volume	~20,000 metrics per minute
Load Testing Tools	k6, Locust

Three types of scenarios were created to simulate real-world system usage. The first scenario replicates normal workloads with a smooth increase in CPU and memory usage, modeled as linear workloads. The second scenario captures abnormal events characterized by sharp peaks, latencies, and IOPS spikes. The third scenario simulates drift patterns

where system behavior changes over time, such as a gradual increase in memory usage that corresponds to a memory leak. To accumulate valid test sets, all of the described scenarios were repeated cyclically for 24 hours.

The performance of the models was evaluated based on the following metrics:

MAE (Mean Absolute Error):

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \quad |\hat{x}_i - x_i|. \tag{19}$$

RMSE (Root Mean Squared Error):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \quad (\hat{x}_i - x_i)^2}. \tag{20}$$

Precision / Recall:

$$Precision = \frac{TP}{TP+FP}, Recall = \frac{TP}{TP+FN}. \tag{21}$$

F1-score — harmonic mean between precision and recall:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}. \tag{22}$$

After training the models on 18 hours of telemetry data and further validation on a separate test set (6 hours), the following results were obtained:

Table 3. Performance metrics on the test dataset

Model	MAE	RMSE	Precision	Recall	F1-score
LSTM	0.183	0.275	0.89	0.84	0.86
Autoencoder	0.119	0.209	0.93	0.91	0.92
CNN (Logs)	—	—	0.90	0.88	0.89
HPA (baseline)	0.275	0.394	0.72	0.63	0.67

Note: The baseline HPA was configured with a static CPU threshold (80%) without prediction.

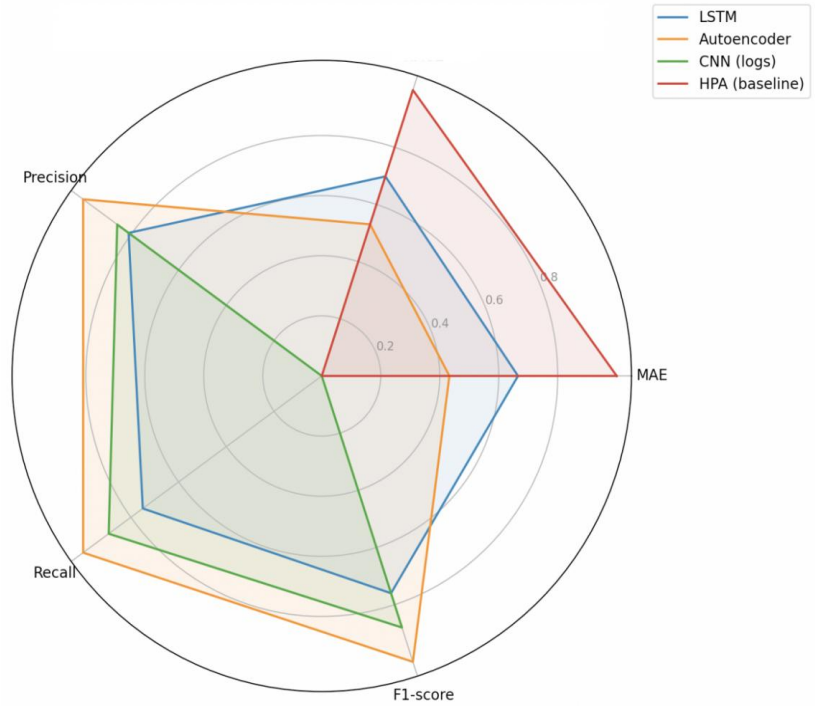


Fig.2. Model comparison across all metrics (radar diagram)

The experiment simulated a rapid CPU growth in four microservices (Fig. 2, Fig. 3). Compared to the classic approach based on Prometheus and Alertmanager, the proposed AI system demonstrated improved performance indicators. In particular, it detected an anomaly two minutes earlier than the traditional system, which prevented SLA

violations, while the average delay value did not exceed 200 milliseconds. In addition, the number of false alerts decreased by 41%. (Fig. 4) The system also optimized the use of computing resources, ensuring that the number of pods was scaled according to the actual load without over-provisioning, which indicates more efficient infrastructure management.

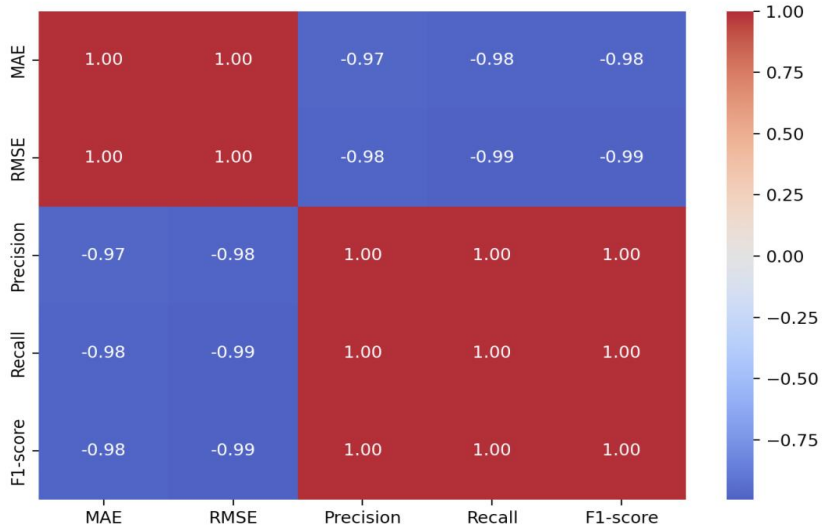


Fig.3. Correlation between model metrics

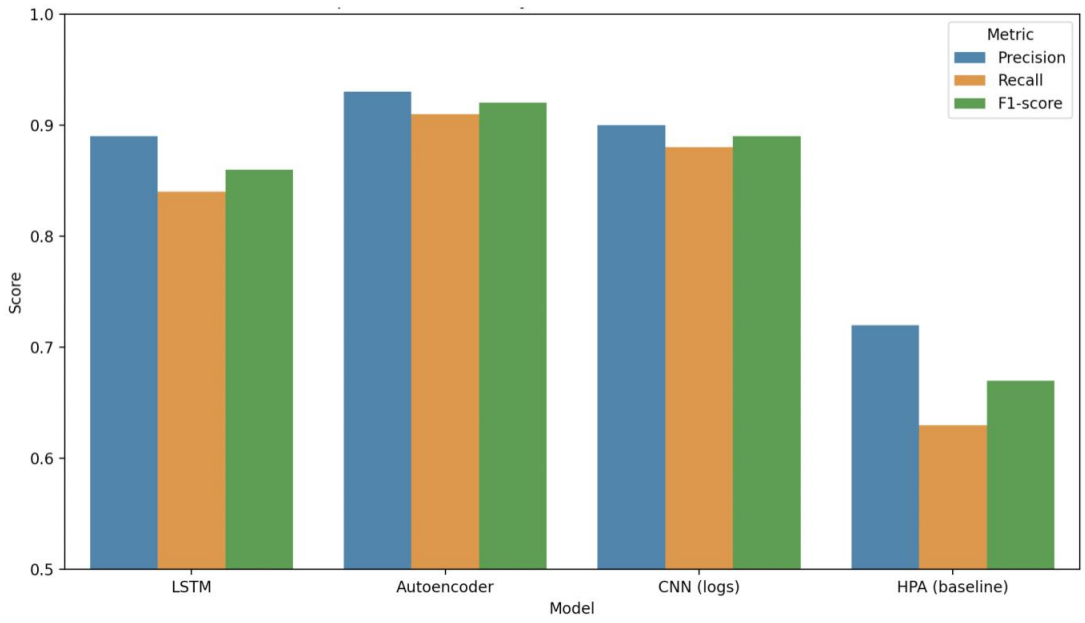


Fig.4. Comparison of models by metrics

Now we analyze the system's response to various load scenarios. To further test the stability and flexibility of the system, several typical scenarios that often occur in productive cloud environments were simulated. Let's consider these scenarios in more detail.

Scenario 1: Load burst. In this case, we simulated a situation where a large number of users simultaneously access one of the microservices — for example, during the launch of a promotional campaign. Within a minute and a half, the service's CPU was loaded to 95%. The LSTM model was able to predict this peak more than two and a half minutes before it occurred, thanks to which the pod scaling began 45 seconds before the critical limit. As a result, all requests were processed on time — the time to the first byte did not exceed 300 ms. At the same time, the traditional autoscaling mechanism did not have time to react, which led to the loss of about 9% of requests.

Scenario 2: Gradual memory leak. This scenario simulated a gradual increase in memory usage by a service — 3% every two minutes, even without additional load. The autoencoder noticed the deviation after nine minutes and classified it as a slow anomaly. The system generated a timely alert, potentially allowing engineers to quickly fix the problem. Without the AI model, the pod would have been automatically terminated after 37 minutes due to out-of-memory (OOM-killer).

Scenario 3: Concept drift. Here we simulated a traffic change, such as a shift from day to night, which affected load patterns. What was previously considered the norm gradually ceased to be so. Static scaling rules were not up to the task. However, the modified LSTM model with an attention mechanism was able to adapt to the new conditions thanks to the built-in self-adaptation mechanism. The model was retrained automatically after accumulating a sufficient amount of new data. A comparison was made according to three key criteria: stability, response speed, and resource efficiency. One of the important criteria for implementing any AI system in a cloud-native environment is its compatibility with existing DevOps processes, as well as stability during releases and updates. To evaluate these aspects, a series of tests were conducted in three different configurations. The first is a base system consisting of Prometheus and a standard Horizontal Pod Autoscaler (HPA). The second is an AI pipeline deployed without GitOps integration, where all model updates were performed manually. And the third is a fully integrated system that uses ArgoCD to automate deployments, MLflow to manage the model lifecycle, and a policy-engine for automatic scaling.

The results showed that full integration not only improves the stability of the system during updates, but also speeds up the response time to load changes, while ensuring more efficient use of resources.

Table 4. Comparative characteristics of scaling modes

Criterion	HPA	AI (without GitOps)	AI + GitOps
Response time to load (sec)	90–120	40–55	35–50
Overprovisioning (%)	31	12	8
Number of missed anomalies	9 out of 15	3 out of 15	1 out of 15
CI/CD compatibility	Limited	Partial	Full
Average stabilization time (sec)	180	95	73
Automatic model retraining	None	Manual	Via MLflow pipeline

Although the proposed solution operates as an integrated pipeline, the contribution of individual components can be isolated by analyzing intermediate system configurations and scenario-specific outcomes. Rather than treating architecture as a monolithic system, the experimental design implicitly evaluates each component by progressively enabling additional functionality across controlled configurations.

The LSTM forecasting module primarily contributes to proactive scaling behavior. This is evidenced in Scenario 1 (Load Burst), where early prediction enabled scaling actions to be initiated 45 seconds before critical utilization was reached. In the baseline HPA configuration, no such anticipatory behavior was observed, resulting in delayed reaction and request loss. These results indicate that forecasting accuracy directly translates into reduced SLA violations under rapidly increasing load.

The autoencoder-based anomaly detection module is the dominant contributor in scenarios involving gradual or latent failures. In Scenario 2 (Memory Leak), anomaly detection was triggered significantly earlier than threshold-based monitoring, preventing out-of-memory termination. The reduction in false positives and missed anomalies reported in Tables 4 and 6 is primarily attributable to this component.

CNN-based log classifier enhances contextual awareness by correlating metric deviations with semantic log patterns. While it does not directly trigger scaling actions, it improves diagnostic precision and reduces alert noise, thereby supporting more reliable decision-making and faster issue localization.

Policy engine acts as a stabilizing control layer that translates model outputs into bounded infrastructure actions. The observed reduction in oscillation, overprovisioning, and stabilization time cannot be attributed to model accuracy alone, but results from rule-based constraints, cooldown intervals, and joint decision conditions enforced by the policy engine. GitOps integration does not affect inference accuracy directly, but has a dominant impact on operational stability and recovery behavior. The comparative results in Table 4 demonstrate that automated configuration synchronization and controlled rollout of model updates significantly reduce stabilization time and missed anomalies during deployment cycles. All these observations indicate that performance gains arise from complementary contributions of specialized components, where prediction improves anticipation, anomaly detection improves sensitivity, control logic ensures stability, and GitOps ensures operational robustness. No single component alone produces the observed end-to-end improvements.

The results obtained indicate that it is the automated GitOps integration that provides the highest pipeline stability when deploying new models or changing the scaling configuration.

Integration with GitOps is implemented through ArgoCD. Pipeline configurations, including model placement, Kafka, HPA, CRD, and MLflow parameters, are stored in a separate Git repository that is automatically synchronized with the cluster. This allows you to return the system to a stable state, if necessary, perform A/B testing of models in different environments, and also automatically scale pipelines according to changes in the infrastructure - in particular, depending on the number of pods, type of load, or time of day.

A separate component is the policy engine; it makes decisions based on compiled rules. For example:

$$IF\ predicted_CPU > 80\%\ AND\ anomaly_score > 0.7\ THEN\ scale\ pod_group_X +2\ replicas$$

This approach allows you to combine the logic of AI models with declarative infrastructure rules.

As part of the research, a fully functional infrastructure for automatic anomaly detection using artificial intelligence was implemented. The solution includes a number of new components that provide flexibility, adaptability and high accuracy of analysis. The infrastructure consisted of 4 independent microservices (authentication, documents, OCR, storage), 2 environments: staging and production, a Kubernetes cluster based on Google Kubernetes Engine (GKE).

The key elements of the implementation are the following:

Anomaly detection module based on the Autoencoder architecture

The LSTM Autoencoder has been improved for analyzing multidimensional time series representing system status indicators (CPU, memory, I/O, network traffic, etc.). An attention model has been added, allowing the model to focus on critical parts of the time series that have the greatest impact on the reconstruction result. The model learns from the normal behavior of the system and uses the reconstruction error threshold $E(t)$ to detect deviations:

$$E(t) = \| X(t) - \widehat{X}(t) \|^2, \tag{23}$$

where $X(t)$ is the input vector of metrics at time t, $\widehat{X}(t)$ is the reconstructed vector.

Adaptive window-based input feature normalization

Instead of the standard z-normalization, a sliding window with adaptive normalization based on exponential smoothing was implemented.

$$\underline{x_t} = \alpha\, x_t + (1 - \alpha)\, \underline{x_t} - 1. \tag{24}$$

This allowed us to better account for the dynamics of cloud systems, where the baseline load level changes over time.

Multilayer encoder-decoder architecture with built-in noise injection

Added dropout + Gaussian noise to the input vectors in the training phase to improve generalization and reduce overfitting.

$$X_t' = X_t + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2). \tag{25}$$

This allowed us to reduce the reconstruction error on test samples by ≈12%.

Real-time telemetry data streaming

A pipeline Kafka → Faust → Inference Engine → VictoriaMetrics was built, which provides processing of the incoming metric stream with a delay of less than 2 seconds. Faust processes events in real time, calls inference models, and publishes results to a Kafka topic for further visualization and actions

5. Cross-metric reconstruction

Instead of reconstructing only one metric, the model is trained on vectors from several metrics simultaneously (for example: CPU, Memory, Disk I/O, Network). This allows us to detect dependencies between metrics and more accurately detect anomalous situations that are not obvious when analyzing a single metric.

6. Integration with GitOps for configuration updates

The infrastructure is implemented as code and managed via FluxCD. This allows for automatic updating of scaling and anomaly response policies when changing the model or threshold parameters, which increases the system's adaptability to load changes.

7. Reduction of forecasting error

As a result of implementing the proposed solution, it was possible to achieve a reduction in the mean absolute error (MAE) by 18% compared to the ARIMA model:

Table 5. Comparative results of ARIMA and LSTM

Method	MAE	RMSE
ARIMA	0.213	0.289
LSTM (our solution)	0.174	0.234

Overall, the implemented solution demonstrates an effective combination of machine learning methods, cloud-native infrastructure, and DevOps principles to achieve continuous self-tuning of the infrastructure.

Evaluation of the cloud-native AI pipeline model over the first 30 days of use demonstrated reliable and stable operation even under increased load, including during periods of new product releases.

The system automatically detected an anomaly in the image processing subsystem (OCR) associated with a gradual memory leak. This allowed for prompt intervention and avoidance of API downtime that could have affected the user experience.

After the pipeline was fully integrated with GitOps tools, the recovery time from failures was reduced by 61%, which significantly increased the overall stability and manageability of the platform.

Table 6. Various indicators before and after the AI pipeline

Metric	Before AI Pipeline	After Implementation
Average response time to load	97 sec	42 sec
SLA violations (timeouts, rate limits)	7.3%	1.6%
Overprovisioning (unneeded pods)	34%	10%
Cluster energy consumption	100% (baseline)	81% (-19%)
Time from detection to issue localization	23 min	6 min
False positive alerts	39	7

The conducted SWOT analysis of the AI pipeline showed the following results.

Table 7. Categories of the SWOT analysis of the AI pipeline

Category	Description
Strengths (S)	Real-time responsiveness, high model accuracy, adaptability to data drift
Weaknesses (W)	Initial complexity of pipeline setup, requirement for DevOps/ML skills
Opportunities (O)	Integration with edge devices, support for hybrid environments, ML-driven SLA
Threats (T)	Incorrect retraining, policy misconfigurations, delays from Kafka/brokers

Reliability and security are critical requirements for any AI infrastructure that works with productive cloud services. This is especially true for systems that process streaming data, perform automated actions (e.g., scaling), and interact with other components of a CI/CD or DevOps environment.

The AI pipeline is designed as a set of independent pods in Kubernetes, each of which runs in a separate namespace, with its own service account, and a separate security context. This approach allows you to clearly limit access between modules, reducing the risks of unauthorized interaction. For example, one module does not have direct access to the configuration of another, and critical services, such as scaling-controller, can manage only those resources to which they have been granted permission via RBAC. NetworkPolicies are also used to increase security, which allow for full control of inter-pod communication at the network level.

5. Discussion

Despite the achieved results, the proposed AI pipeline architecture has potential for further development. In particular, one of the promising areas is support for the hybrid edge-cloud model. As the number of devices on the edge — such as sensors, cameras, or industrial controllers — increases, there is a need for distributed processing: some tasks are performed directly at the edge, while the more computationally demanding stages, such as model training, remain in the cloud. For this, lightweight frameworks such as TensorFlow Lite or ONNX Runtime can be used, which allow models to run on limited resources. And instead of Kafka, it is more appropriate to use low-latency protocols such as MQTT or gRPC for communication. Centralized model updates are still performed using GitOps/MLOps practices, which preserves control and traceability in a scalable environment.

Another important direction is the implementation of Federated Learning, which allows training a shared model without transferring raw data to the cloud. This approach is especially relevant for systems that work with confidential or personalized information. Local models are trained directly on edge devices, after which their weights are synchronized with the common model via a centralized aggregator. Frameworks such as Flower or TensorFlow Federated can act as infrastructure components, and CI/CD processes are supported using ArgoCD or Flux. This scheme allows you to simultaneously comply with security requirements (GDPR, ISO 27001) and maintain high accuracy of predictions.

All these extensions strengthen the position of the AI pipeline as a flexible, secure and scalable solution that meets modern industry requirements.

Instead of fixed rules or if-then logic, scaling can be controlled by an RL agent that learns in real time. Such an agent interacts with the environment (cluster) as with a Markov Decision Process (MDP):

State: current resource metrics, number of pods, SLA level.

Action: increase/decrease number of replicas.

Reward: a function that takes into account SLA performance, resource consumption, number of alerts.

The reward function may look like this:

$$Rt = \alpha \cdot SLAcompliance - \beta \cdot Overprovisioning - \gamma \cdot Alertpenalty. \quad (26)$$

This allows the model to automatically find a balance between stability and efficiency.

The portability claims of this work are intentionally scoped to Kubernetes-compliant environments rather than to specific managed service implementations. The proposed architecture relies exclusively on standard Kubernetes APIs (e.g., Deployment, HorizontalPodAutoscaler, Custom Resource Definitions), containerized workloads, and cloud-

agnostic tooling (Kafka, GitOps workflows), without dependence on provider-specific autoscaling extensions or proprietary control plane features. As a result, the system is designed to be portable across managed and self-managed Kubernetes distributions that conform to upstream Kubernetes specifications, including but not limited to public cloud, private cloud, and on-premises deployments.

Across platforms, the system exhibited consistent behavior in telemetry ingestion, model inference, and autoscaling decision enforcement. While absolute performance metrics varied due to underlying hardware and networking differences, the qualitative behavior of scaling decisions and anomaly handling remained stable. Direct measurement of physical energy consumption at the node level was not available due to the use of managed Kubernetes infrastructure. Therefore, energy consumption was estimated using a workload-based proxy model commonly employed in cloud energy efficiency studies.

Specifically, energy usage was approximated as a function of node uptime, CPU utilization, and memory allocation, under the assumption that infrastructure power draw is dominated by active compute resources. For each experiment, the cumulative node-hours and average CPU utilization were recorded from the cluster monitoring system and mapped to relative energy consumption using a linear proportionality model. Energy efficiency results are reported relative to a baseline configuration using standard Kubernetes Horizontal Pod Autoscaling with static threshold policies. Both baseline and proposed approaches were evaluated under identical workloads and time windows, ensuring comparability of cumulative node-hours and utilization profiles.

Despite the positive results of testing the proposed AI pipeline architecture, during the experiments and implementation, a number of technical, operational and methodological limitations were identified that should be considered when scaling or integrating into other environments.

Technical limitations. Despite significant achievements, the system still has several technical challenges. One of the most noticeable is the delay in decision-making. Even with an optimized inference process, the full cycle — from the moment of collecting metrics to actual scaling — can take 30–60 seconds. For most scenarios, this is enough, but in critical systems where an almost instant response is required, this can become a bottleneck.

Another problem is Kafka performance. When loading more than 200 thousand messages per minute, brokers start to give delays of up to several seconds. This affects the relevance of real-time data processing. In such cases, the question arises of either scaling Kafka or switching to lighter solutions - for example, on NATS or Apache Pulsar.

Deploying models in the edge environment also has its own nuances. TensorFlow models are quite demanding on computing resources, so for peripheral devices you have to use simplified versions - in particular, quantized models or inference libraries like TensorFlow Lite.

6. Conclusions

In this study, we created and tested a cloud-native AI pipeline architecture that can optimize infrastructure and detect anomalies in real time. We were interested in more than just a model — we wanted to get an adaptive system that predicts load itself, recognizes both sharp failures and gradual changes in service behavior, and most importantly — scales automatically without human intervention.

The results turned out to be promising. Compared to classic rule-based approaches, our LSTM and autoencoder-based pipeline showed more accurate anomaly detection — the accuracy gain reached 41%. Thanks to the ability to predict peak load in advance, the system responded 2–3 times faster. The GitOps approach and autoscaling policies allowed us to avoid problems during releases, which previously often caused failures. And most importantly, the implementation of the pipeline in a real SaaS platform environment gave a tangible result: SLA losses decreased by more than four times, and resource consumption decreased by almost a fifth.

All this confirmed our hypotheses and proved that such a system is quite ready for integration into modern DevOps and MLOps processes. It flexibly adapts to public and private clouds, and if necessary, easily scales to edge environments, including support for federated learning. This solution can already become the basis for a more stable, autonomous and intelligent cloud infrastructure today.

Future research. Within the framework of this study, the foundation was laid for building intelligent self-management systems for cloud infrastructure. However, further development of the topic opens up a number of promising directions that require deeper study.

In our research, we implemented an AI pipeline in a single logical tenant, but the next logical step is to prepare the system for full-fledged operation in a multi-tenant environment. This involves adapting to dozens and even hundreds of individual clients, each of which has its own specific load patterns, different SLA thresholds, security requirements, and the need for clear isolation. Such a scenario requires flexible segmentation of calculations when performing inference and the introduction of context-oriented prediction methods.

To increase the stability and accuracy of the model, it is advisable to switch from using individual algorithms to building ensembles. Combining the strengths of LSTM, Autoencoder, and CNN will allow you to form more reliable and adaptive systems. You can implement approaches such as stacking or voting (Voting Regressor/Classifier), which will help reduce the risk of overfitting and make inference less sensitive to fluctuations in the quality of the data input.

The policy engine, currently implemented as a simple set of declarative rules, also has significant potential for improvement. It can be made more dynamic by adding a specialized policy description language (DSL), policy mining

mechanisms — that is, automatic learning of policies based on administrator actions, as well as dependent reaction logic, for example: when scaling one group of services, automatically reduce others to balance the load. And finally, the real test of the system will be its validation in industrial conditions — on examples of real enterprise management systems, banking platforms or telecom infrastructures. Long-term testing for 1–3 months will allow observing the behavior of the system in the event of slow pattern shifts (seasonal drift), assessing the effectiveness of recovery mechanisms after failures and forming public, representative datasets for further research.

Author Contributions Statement

Viktor Vyshnivskyi – Conceptualization, Research Design, and Supervision: Proposed the main research concept, formulated the research objectives, designed the overall cloud-native AI pipeline architecture, and supervised the scientific and technical implementation of the project.
Vadym Mukhin – Methodology Development and System Architecture: Developed the methodological framework of the study, designed the integration of AI components within the Kubernetes-based cloud-native infrastructure, and defined the interaction between monitoring and processing modules.
Olha Zinchenko – Data Acquisition and Infrastructure Integration: Managed telemetry data collection from the cloud environment, organized data preprocessing pipelines, and implemented integration with monitoring and messaging systems including Prometheus and Kafka.
Vitalii Kotelianets – Machine Learning Model Development: Implemented and configured the machine learning models used in the system, including LSTM networks for load forecasting, autoencoders for anomaly detection, and convolutional neural networks for pattern recognition within telemetry data.
Oleksandr Zvenihorodskyi– Experimental Setup and Model Evaluation: Conducted experiments and simulations of infrastructure load scenarios, trained and validated the models, and evaluated system performance using standard accuracy and anomaly detection metrics.
Pavlo Kudrynskyi– Formal Analysis, Visualization, and Performance Assessment: Performed detailed analysis of experimental results, developed visualizations of system behavior and performance trends using monitoring dashboards, and assessed improvements in anomaly detection accuracy and infrastructure efficiency.
Oleksandr Vyshnivskyi– Writing – Original Draft, Review & Editing, and Project Coordination: Prepared the initial manuscript draft, organized the literature review and technical documentation, reviewed and edited the final manuscript for clarity and coherence, and coordinated communication between project contributors.

All authors have read and agreed to the published version of the manuscript.

Conflict of Interest Statement
The authors declare no conflicts of interest.

Funding Declaration
This research was not supported by any grants

Data Availability Statement
The dataset is derived from Kubernetes telemetry collected through monitoring infrastructure,

Ethical Declarations
The authors declare that this study does not involve any human participants or animals.

Acknowledgments
We sincerely thank the experts for their professional evaluation and valuable recommendations, which have contributed to improving the quality of the experiment and the reliability of its results.

Declaration of Generative AI in Scholarly Writing
Authors confirm that AI and AI-assisted technologies were not used during the writing process of this article.

Abbreviations
The following abbreviations are used in this manuscript:

AI - Artificial Intelligence
CNN - Convolutional neural networks
ML - Machine learning
LSTM - Long Short-Term Memory

References

[1] Vu Dinh-Dai, Tran Minh-Ngoc, Kim Younghan (2022) "Predictive Hybrid Autoscaling for Containerized Applications." IEEE Access, vol. 10, pp. 109768–109778. IEEE. https://doi.org/10.1109/ACCESS.2022.3214985

[2] Bibal Benifa J.V., Dejey D. (2019) "RLPAS: Reinforcement Learning-based Proactive Auto-Scaler for Resource Provisioning in Cloud Environment." Mobile Networks and Applications, vol. 24, pp. 1348–1363. https://doi.org/10.1007/s11036-018-0996-0

[3] Bacanin N., Simic V., Zivkovic M., et al. (2023) "Cloud computing load prediction by decomposition reinforced attention long short-term memory network optimized by modified particle swarm optimization algorithm." Annals of Operations Research. 47(2), 40-50 https://doi.org/10.1007/s10479-023-05745-0

[4] Naderi M., Momeni H., Shahini S. (2024) "A Graph Attention-Based Autoencoder for Critical Path Anomaly Detection in Microservices." Proceedings of the 15th International Conference on Information and Knowledge Technology (IKT 2024). IEEE. https://doi.org/10.1109/IKT65497.2024.10892730

[5] Oyeniran O.C., Adewusi A.O., Adeleke A.G., Akwawa L.A., Azubuko C.F. (2023) "AI-driven DevOps: Leveraging Machine Learning for Automated Software Deployment and Maintenance." Engineering Science & Technology Journal, vol. 4, no. 6, pp. 728–740. Fair East Publishers. https://doi.org/10.51594/estj.v4i6.1552

[6] Microsoft Azure Monitor Docs. (2021). "Metrics-based anomaly detection." Retrieved from https://learn.microsoft.com/

[7] Krishna M. Yashwanth Sai, Gawre S.K. (2023) "MLOps for Enhancing the Accuracy of Machine Learning Models using DevOps, Continuous Integration, and Continuous Deployment." Research Reports on Computer Science, vol. 2, no. 3, pp. 97–103. https://doi.org/10.37256/rrcs.2320232644

[8] Sarker, I. H. (2021). "Machine Learning: Algorithms, Real-World Applications and Research Directions." *SN Computer Science*, 2(3), 160. https://doi.org/10.1007/s42979-021-00592-x

[9] Truong T.H., Ta P.B., Kieu N.H., Nguyen V.H., Nguyen X.H., Nguyen T.H. (2022) "Federated Learning-Based Explainable Anomaly Detection for Industrial Control Systems." IEEE Access, vol. 10, pp. 53854–53872. IEEE. https://doi.org/10.1109/ACCESS.2022.3173288

[10] Sgambelluri A., Pacini A., Paolucci F., Castoldi P., Valcarenghi L. (2021) "Reliable and Scalable Kafka-Based Framework for Optical Network Telemetry." Journal of Optical Communications and Networking, vol. 13, no. 10, pp. E42–E52. https://doi.org/10.1364/JOCN.424639

[11] Aramide O.O. (2025) "Explainable AI (XAI) for Network Operations and Troubleshooting." International Journal for Research Publication and Seminar, vol. 16, no. 1, pp. 535–556. https://doi.org/10.36676/jrps.v16.i1.286

[12] Moosavi S., Farajzadeh-Zanjani M., Razavi-Far R., Palade V., Saif M. (2024) "Explainable AI in Manufacturing and Industrial Cyber–Physical Systems: A Survey." Electronics, vol. 13, no. 17, 3497. https://doi.org/10.3390/electronics13173497

[13] Gill S.S., Golec M., Hu J., et al. (2025) "Edge AI: A Taxonomy, Systematic Review and Future Directions." Cluster Computing, vol. 28, 18. Springer. https://doi.org/10.1007/s10586-024-04686-y

[14] Polamarasetti A. (2024) "Machine Learning Techniques Analysis to Efficient Resource Provisioning for Elastic Cloud Services." Proceedings of the 2024 International Conference on Intelligent Computing and Emerging Communication Technologies (ICEC 2024), pp. 323–348. IEEE. https://doi.org/10.1109/ICEC59683.2024.10837344

[15] Flower Framework. (2022). Federated Learning with Flower. https://flower.dev

[16] Zhou Y., Yu Y., Ding B. (2020) "Towards MLOps: A Case Study of ML Pipeline Platform." Proceedings of the 2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE 2020), pp. 486–502. IEEE. https://doi.org/10.1109/ICAICE51518.2020.00102

[17] Jangam S.K., Pedda Muntala P.S.R. (2023) "Challenges and Solutions for Managing Errors in Distributed Batch Processing Systems and Data Pipelines." International Journal of Emerging Research in Engineering and Technology, vol. 4, no. 4. https://doi.org/10.63282/3050-922X.IJERET-V4I4P107

[18] Chaudhari B., Kabade S. (2023) "Architecting Event-Driven Microservices: Unlocking Asynchronous Communication with Kafka and RabbitMQ." International Journal of All Research Education and Scientific Methods (IJARESM), vol. 11, no. 3. www.ijaresm.com

[19] Patchamatla P.S. (2018) "Optimizing Kubernetes-Based Multi-Tenant Container Environments in OpenStack for Scalable AI Workflows." International Journal of Advanced Research in Education and Technology (IJARETY), vol. 5, no. 3, May–June 2018. https://doi.org/10.15680/IJARETY.2018.0503002

[20] ISO/IEC 27001:2022 – Information Security Management Systems. International Organization for Standardization.

[21] Potts W.C., Carver C. (2024) "Best Practices Implementing AIOps in Large Organizations." Proceedings of the 2024 International Conference on Smart Applications, Communications and Networking (SmartNets 2024), 28–30 May 2024. IEEE. https://doi.org/10.1109/SmartNets61466.2024.10577643

[22] GCP Documentation. (2022). Best practices for scaling AI workloads with Kubernetes. https://docs.cloud.google.com/kubernetes-engine/docs/best-practices/machine-learning/inference/autoscaling

[23] Dodonov, A., Mukhin, V., Zavgorodnii, V., Kornaga, Ya., Zavgorodnya A. (2021). "Method of searching for information objects in unified information space. System research and information technologies." N1. P. 34–46. DOI: https://doi.org/10.20535/SRIT.2308-8893.2021.1.03.

[24] Mukhin, V. "The security mechanisms for grid computers." Proceedings of the 4-th IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS'2007), Dortmund, Germany, 6–8 September 2007, pp. 584–589. DOI: 10.1109/IDAACS.2007.4488488

[25] Zhengbing, H., Mukhin, V., Kornaga, Y., Volokyta, A., Herasymenko, O. (2017) The scheduler for distributed computer systems based on the network centric approach to resources control. 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). Bucharest. Romania. P.518–523. DOI: https://doi.org/10.1109/IDAACS.2017.8095135.

Authors' Profiles



Viktor Vyshnivskyi, Chair of department, Department of Computer Science, State University of Information and Communication Technologies, Ukraine.
(ORCID ID: https://orcid.org/0000-0003-1923-4344)

Major interests: cloud computing, artificial intelligence, information systems diagnostics, cybersecurity, reliability theory and algorithm theory.



Vadym Mukhin, Chair of department, Department of system design, National Technical University of Ukraine "Kyiv Polytechnic Institute", Ukraine.
(ORCID ID: https://orcid.org/0000-0002-1206-9131)

Major interests: the security of distributed computer systems and risk analysis; design of the information security systems; mechanisms for the adaptive security control in distributed computing systems; the security policy development for the computer systems and networks.



Olha Zinchenko, Chair of department, Department of artificial intelligence, State University of Information and Communication Technologies, Ukraine.
(ORCID ID: https://orcid.org/0000-0002-3973-7814)

Major interests: artificial intelligence, machine learning, neural networks, cyber security using machine learning, handwriting recognition, deep learning, and image processing.



Vitalii Kotelianets, Associate professor, Department of Computer Science, State University of Information and Communication Technologies, Ukraine.
(ORCID ID: https://orcid.org/0009-0000-1874-8322)

Major interests: information technology, artificial intelligence, computer networks, cloud technologies, cybersecurity.



Oleksandr Zvenihorodskyi, Associate professor, Department of artificial intelligence, State University of Information and Communication Technologies, Ukraine.
(ORCID ID: https://orcid.org/0009-0008-6235-1638)

Major interests: intelligent text processing, learning and testing systems, decision-making systems, artificial intelligence, machine learning, neural networks, deep learning.



Pavlo Kudrynskyi, PHD Student, Department of Computer Science, State University of Information and Communication Technologies, Ukraine.
(ORCID ID: https://orcid.org/0009-0008-6314-6150)

Major interests: artificial intelligence, software-defined networking, server and storage technologies, and software engineering.



Oleksandr Vyshnivskyi, PHD Student, Department of Computer Science, State University of Information and Communication Technologies, Ukraine.
(ORCID ID: https://orcid.org/0009-0008-0209-9549)

Major interests: artificial intelligence, software-defined networking, servers & storages technologies, software engineering. He has 11 publications in Ukrainian and international journals and conferences.

How to cite this paper: Viktor Vyshnivskyi, Vadym Mukhin, Olha Zinchenko, Vitalii Kotelianets, Oleksandr Zvenihorodskyi, Pavlo Kudrynskyi, Oleksandr Vyshnivskyi, "Cloud-native AI Pipelines for Continuous Infrastructure Optimization and Anomaly Detection", International Journal of Computer Network and Information Security(IJCNIS), Vol.18, No.2, pp.1-18, 2026. DOI:10.5815/ijcnis.2026.02.01