A Runtime Anomaly Detection and Fixing Framework Using LLM-Powered Separate guided Graph Attention Recurrent Compact Conviformer for Web Applications

Partha Pratim Biswas[1*], **Dr. Amit Dixit**[1], **Dr. Tanupriya Choudhury**[2]

[1]Department of Computer Science and Engineering, Quantum University, Roorkee - 247167, Uttarakhand, India

[2]Department of Computer Science and Engineering, UPES, P.O, Bidholi Via, Prem Nagar, Dehradun - 248007, Uttarakhand, India

[1]PhD Scholar (CSE) (Partha Pratim Biswas) | Email: ppbiswas82@gmail.com

[1]Professor (Dr. Amit Dixit) | Email: amitdixit.ece@quantumeducation.in

[2]Research Professor (Dr. Tanupriya Choudhury) | Email: tanupriya1986@gmail.com

***Corresponding Author:** Partha Pratim Biswas, PhD Scholar (CSE), Department of Computer Science and Engineering, Quantum University, Roorkee - 247167, Uttarakhand, India | Email: ppbiswas82@gmail.com

Abstract

Modern web applications, particularly microservice-based, face numerous runtime anomalies from unexpected inputs, dynamic routing, permission conflicts, and misconfigured services. Existing detection systems also lack real-time adaptability and automated fixes, requiring manual intervention and reducing system resilience. To address these limitations, this research proposes a comprehensive runtime anomaly detection and fixing framework for Python web applications built on FastAPI. It employs a Separate guided Graph Attention Recurrent Compact Conviformer (SGARCC), which integrates graph-based modelling with recurrent attention mechanisms, enabling the detection of long-range temporal dependencies and relational patterns in log sequences. Hyperparameter optimization is achieved through the Bighorn Sheep Optimization Algorithm (BSOA), which dynamically tunes learning rates, dropout probabilities, and attention thresholds to enhance accuracy and reduce inference latency. Upon anomaly detection, Local Interpretable Model-agnostic Explanations (LIME) generate feature-level visualizations to improve transparency. large language models, specifically Gemma3, then provide context-aware, semantically correct code fixes, leveraging error logs, impacted code, and historical corrections. The proposed SGARCC framework achieved a remarkable accuracy of 97.03%, precision of 95.57%, recall of 97.03%, and F1-score of 96.30%, demonstrating superior effectiveness in runtime anomaly detection and automated remediation for web applications. The SGARCC framework provides a real-time, self-healing solution for Python web applications by integrating graph-attention recurrent modelling, optimized hyperparameters, LIME-based interpretability, and LLM-driven automated code fixes, demonstrating superior effectiveness in runtime anomaly detection and remediation.

Keywords: *Web applications, Bighorn Sheep Optimization Algorithm, Local Interpretable Model-Agnostic Explanations, Automated Remediation, Large Language Models.*

How to cite this article: Biswas PP, Dixit A, Choudhury T. A runtime anomaly detection and fixing framework using LLM-powered separate guided graph attention recurrent compact conviformer for web applications. Int J Drug Deliv Technol. 2026;16(75s): 1829-1844. DOI: 10.25258/ijddt.16.75s.195

1. Introduction

The increasing complexity of web applications has transformed the digital ecosystem, offering advanced interactivity, scalability, and real-time responsiveness. However, this evolution has also introduced significant challenges in maintaining system reliability during execution [1]. Runtime anomalies such as latency fluctuations, memory leaks, thread deadlocks, and unexpected system crashes severely impact user experience, degrade performance, and compromise service availability [2]. These anomalies often arise due to intricate dependencies among distributed services, asynchronous communication, dynamic APIs, and heterogeneous workloads that evolve continuously under varying operational conditions. Conventional anomaly detection systems primarily rely on rule-based monitoring, statistical modelling, or shallow learning mechanisms, which lack adaptability and contextual understanding [3]. Such approaches struggle to handle dynamic runtime behaviours, especially in large-scale, data-driven, and decentralized environments where execution patterns

constantly change. Moreover, they often fail to generalize across different deployment contexts, leading to high false alarm rates and delayed detection [4]. The absence of intelligent mechanisms for autonomous correction further complicates the reliability of modern web infrastructures. In recent years, advances in Deep Learning (DL) and Artificial Intelligence (AI) have paved the way for more sophisticated runtime anomaly detection techniques [5]. Particularly, the integration of graph-based representations and attention-driven models has enabled the exploration of structural and temporal correlations within runtime execution data. These models effectively identify hidden dependencies, non-linear relationships, and contextual variations that contribute to anomalous behaviours [6]. Simultaneously, the emergence of Large Language Models (LLMs) has introduced novel capabilities for semantic reasoning, adaptive learning, and self-correction by understanding both structured system data and unstructured log semantics [7]. Within this evolving landscape, there is a growing necessity for frameworks that combine contextual awareness, dynamic adaptability, and interpretability to ensure reliable anomaly detection and response in web applications [8]. This research serves as a continuation of the earlier published paper reported in [9], which introduced a foundational framework for adaptive anomaly detection in microservice-driven web systems using AI-assisted decision logic. The previous research primarily focused on the conceptual integration of runtime monitoring and shallow learning-based anomaly identification to enhance operational stability in dynamic application environments. However, while the initial approach demonstrated the feasibility of intelligent runtime observation, it lacked advanced temporal–relational modelling and autonomous remediation capabilities. Building upon that groundwork, the present research advances the framework significantly by incorporating an SGARCC architecture that captures both structural and temporal dependencies within runtime log sequences. Furthermore, an LLM-powered module has been integrated to enable context-aware, semantically precise automated code correction, transforming the system from passive anomaly detection to a fully self-healing runtime environment. These improvements address the scalability, interpretability, and adaptability limitations of the earlier framework, thereby establishing a comprehensive solution for real-time anomaly detection and automated remediation in FastAPI-based web applications. Thus, this research builds upon the foundations of the initial research.

1.1. Contributions

- This research integrates interpretable anomaly explanations and automated remediation by employing LIME for feature-level transparency and a LLM, Gemma3, for context-aware code correction. By generating semantically correct patches without human intervention, the framework enables self-healing web applications, reduces downtime, and enhances trustworthiness, bridging the gap between detection and actionable correction in runtime anomaly management.
- This research introduces the SGARCC model, combining graph-based attention and recurrent mechanisms to capture long-range temporal dependencies and relational patterns in application logs. Optimized with the BSOA, the framework dynamically tunes hyperparameters, achieving high accuracy, robustness, and low-latency inference in noisy, real-world runtime environments.
- This research proposes an end-to-end runtime anomaly detection framework tailored for Python-based FastAPI web applications. By leveraging ongoing server monitoring and logging, it captures semantically rich, time-ordered data to detect diverse runtime anomalies that often evade static analysis or unit testing, thereby improving the reliability and resilience of microservice-based web systems.

1.2. Literature review

Traditional approaches, including static code analysis and unit testing, are largely limited to pre-deployment environments, failing to capture anomalies arising from dynamic user inputs, misconfigured services, or runtime permission conflicts. To overcome these limitations, recent research has explored the integration of DL and LLMs to enable adaptive, automated self-healing systems. **Xu et al. [10]** proposed a two-staged LLM-based framework for CI/CD pipeline failure detection and remediation. The first stage employed log embeddings and anomaly classification to identify failing builds, while the second stage leveraged LLMs for generating context-aware remediation suggestions. Industrial validation demonstrated reductions in build failures and manual debugging efforts, confirming the feasibility of LLM-driven automated interventions. **Toprani and Madisetti [11]** designed an LLM agentic workflow for automated vulnerability detection and remediation in Infrastructure-as-Code (IaC). The workflow combined semantic code analysis with multi-agent coordination, enabling autonomous patch generation and outperforming traditional static analyzers. **Yonathan [12]** presented RuntimeErrorSage, which utilized local LLMs for real-time detection and correction of runtime errors in Python applications, emphasizing

low-latency remediation and data privacy. Table 1 shows the summary comparison of anomaly detection.

Table 1: Comparative Analysis of Anomaly Detection Frameworks

Refs	Models	Architecture	Remediation	Interpretability	Data Source
[10]	LLM-Based CI/CD	Two-stage LLM	Yes	Partial	Industrial CI/CD logs
[11]	LLM Agentic	LLM Agentic Workflow	Yes	Partial	IaC scripts/ logs
[12]	RuntimeErrorSage	Local LLM	Yes	No	Web app runtime logs
[13]	Pre-trained LLMs	Pre-trained LLM	Limited	No	Declarative specification datasets
[14]	FIRED	Robust performance	No	Partial	Cloud runtime logs
[15]	TPAD	Temporal-pattern-based NN	No	No	Multivariate time-series logs
[16]	ADAL-NN	Deep relational NN	No	No	Distributed system logs
[17]	AADRF	AI-based anomaly detection	Yes	No	Firmware runtime logs
[18]	Argos	LLM + autonomous rule generation	Yes	Partial	Time-series / cloud logs
[19]	HGAIAM	AI-based anomaly detection	No	Yes	Cloud ML logs and API datasets
[20]	ISMS	Temporal-pattern-based NN	Partial	Yes	Server metrics
[21]	LLMALHD	LLM + autonomous rule generation	Yes	No	Network intrusion datasets
Proposed SGARCC		Graph Attention + Recurrent Conviformer + LLM	Yes	Yes (LIME)	Web server runtime logs (Fast API)

Alhanahnah et al. [13] conducted an empirical evaluation of pre-trained LLMs for repairing declarative formal specifications, focusing on their ability to understand and correct domain-specific constraints. The study highlighted both the potential of LLMs in automating specification repair and their limitations in handling highly formalized or context-dependent reasoning, emphasizing the need for fine-tuning and integration with domain knowledge to improve reliability. **Xin et al. [14]** proposed a fine-grained robust performance diagnosis framework for run-time cloud applications, combining distributed tracing, causal inference, and statistical analysis to accurately identify performance bottlenecks at a service or microservice level. **Ma et al. [15]** introduced Temporal-Pattern-based neural network for Anomaly Detection (TPAD), a deep learning model designed for multivariate time-series data. By leveraging temporal patterns and correlations across multiple variables, TPAD effectively detects subtle anomalies in sensor networks and industrial monitoring systems, outperforming conventional statistical and sequence-based methods in both accuracy and detection latency. Similarly, **Otieno [16]**

introduced an AI-enhanced Anomaly Detection and autonomous Remediation Framework (AADRF) for Unified Extensible Firmware Interface Basic Input/Output System (UEFIBIOS) firmware, combining behavioral profiling with autonomous correction agents. The system detected low-level firmware anomalies and generated secure corrective actions, improving resilience in heterogeneous computing environments. **Ahmed et al. [17]** proposed Anomaly Detection and Advanced Localization using Neural Networks (ADAL-NN), leveraging deep relational learning to localize anomalies across distributed systems. The model captures complex interdependencies among system components by representing them as relational graphs, allowing it to identify not only the occurrence of anomalies but also their exact location within the distributed network. **Gu et al. [18]** introduced Argos, which combined agentic LLMs and autonomous rule generation for adaptive time-series anomaly detection, achieving high interpretability and adaptive rule synthesis. **Tran [19]** proposed a Hybrid Generative AI–driven Automation Model (HGAIAM) designed for intelligent vulnerability detection and remediation in cloud-based machine learning systems. The model integrated generative artificial intelligence with adaptive automation pipelines to proactively identify vulnerabilities arising from data drift, model misconfigurations, and insecure API endpoints. **Luis et al. [20]** developed an Intelligent Server Monitoring System (ISMS) that utilized multiple machine learning algorithms for real-time anomaly detection and automated response. Upon detection, the framework automatically triggered corrective actions such as process restarts, load balancing, or service isolation, thereby improving uptime and ensuring high service availability without human intervention. **Luo et al. [21]** introduced LLM-enabled Active Learning for Human-free anomaly Detection (LLMALHD), a next-generation framework employing large language models to facilitate autonomous network anomaly detection. By leveraging contextual reasoning and semantic understanding capabilities of LLMs, LLMALHD enhanced detection precision in complex, high-dimensional network environments and demonstrated superior adaptability in dynamic threat landscapes.

1.3. Research gaps

Despite the promising results achieved by the SGARCC framework in enhancing runtime anomaly detection and autonomous remediation in web applications, several research gaps remain that warrant further investigation. Additionally, while the integration of LLMs enables intelligent self-healing, the dependency on external inference services introduces latency or privacy risks in highly sensitive or resource-constrained systems. The model's scalability under extreme traffic loads, evolving anomaly patterns, or adversarial conditions also requires deeper examination to ensure consistent robustness. Furthermore, explainability is demonstrated through LIME-based interpretations at the token level, yet a comprehensive causal interpretability framework for complex anomaly dependencies remains absent. Addressing these limitations paves the way for more secure, adaptive, and transparent autonomous web systems in future research.

1.4. Research Questions and Hypotheses

⇒ **RQ1:** How effectively does the SGARCC model detect and classify runtime anomalies in web applications, and does the integration of the BSOA enhance its convergence efficiency and inference speed in real-time runtime monitoring scenarios?

H1: The SGARCC model achieves higher anomaly detection accuracy and lower false positive rates than traditional models such as LSTM, CNN, or Transformer-based architectures, due to its integrated graph-attention and recurrent compact representation that captures both relational and temporal dependencies within runtime logs. Additionally, incorporating the BSOA significantly improves the convergence rate, optimizes hyperparameters dynamically, and reduces inference latency, enabling real-time anomaly detection with minimal computational overhead.

⇒ **RQ2:** To what extent does LLM-powered contextual reasoning enable autonomous and semantically accurate remediation of detected runtime anomalies in Python-based web applications?

H2: The integration of LLM-based reasoning (Gemma3) enables the framework to generate context-sensitive, syntactically valid, and semantically coherent code fixes, leading to reduced recovery time and improved system resilience compared to rule-based or manually supervised remediation methods.

1.5. Organization

The research is organized as follows: Section 1 introduces the study's motivation and objectives; Section 2 reviews related literature; Section 3 describes the proposed SGARCC methodology; Section 4 presents results and performance evaluations; Section 5 discusses findings, limitations, and implications; and Section 6 concludes with key outcomes and future research directions.

2. Proposed methodology

The proposed methodology presents an end-to-end runtime anomaly detection and remediation framework for Python-based FastAPI web applications. It begins with continuous monitoring and logging of server activity to capture time-ordered, semantically rich data reflecting application behavior. The framework employs the SGARCC, which integrates graph-based attention and recurrent mechanisms to model relational and temporal dependencies in log sequences, enabling accurate detection of diverse runtime anomalies. Hyperparameters, including learning rate, dropout probabilities, and attention thresholds, are dynamically optimized using the BSOA to ensure high accuracy and low-latency inference. Detected anomalies are classified, such as syntax errors, permission violations, or null pointer exceptions. LIME provides transparent feature-level insights. For automated correction, the framework leverages the Gemma3 LLM to generate context-sensitive and semantically correct code patches, facilitating self-healing and reducing downtime.

2.1. Monitoring and Logging

The proposed framework begins with a comprehensive monitoring and logging mechanism to capture the runtime behaviour of a Python web application built using FastAPI, with PostgreSQL as the backend database and Psycopg as the database adapter. A detailed workflow of FastAPI–PostgreSQL anomaly detection and automated remediation is shown in Figure 1.

This setup allows continuous collection of time-stamped, semantically rich, and relational data from the application, including HTTP request-response cycles, database queries, execution traces, and runtime errors. Monitoring is implemented as a middleware in FastAPI, intercepting every incoming request and outgoing response. Each log entry records critical metadata, including timestamp, endpoint, HTTP method, request payload, response status, execution latency, and database interactions. These logs form a structured dataset that captures both temporal sequences and inter-event dependencies, which are essential for detecting anomalies in real-time.

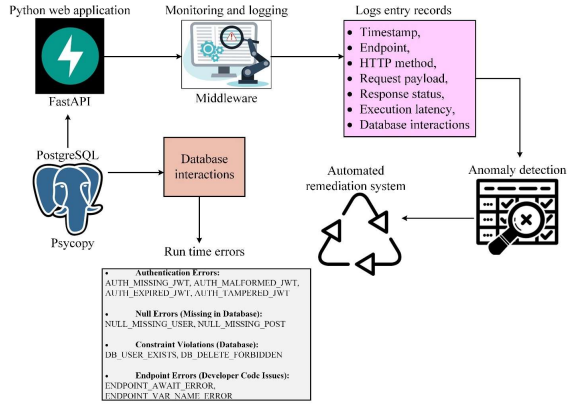


Figure 1. Architectural flow of the proposed anomaly detection and automated remediation framework

The framework identifies and records various types of errors that occur during runtime, including:

- **Authentication Errors:** AUTH_MISSING_JWT, AUTH_MALFORMED_JWT, AUTH_EXPIRED_JWT, AUTH_TAMPERED_JWT
- **Null Errors (Missing in Database):** NULL_MISSING_USER, NULL_MISSING_POST
- **Constraint Violations (Database):** DB_USER_EXISTS, DB_DELETE_FORBIDDEN
- **Endpoint Errors (Developer Code Issues):** ENDPOINT_AWAIT_ERROR, ENDPOINT_VAR_NAME_ERROR

These logs serve as the input for the anomaly detection model. By analyzing patterns in these error sequences, the model identifies runtime anomalies, which are subsequently corrected by the automated remediation system. This enables real-time detection and resolution, minimizing downtime and maintaining reliable application performance.

2.2. Separate guided Graph Attention Recurrent Compact Conviformer for classification

Following the monitoring and logging step, the structured log sequences, including various runtime errors, are fed into the anomaly detection model. Traditional sequence models often struggle to capture temporal dynamics and relational dependencies in log data simultaneously. To overcome this limitation, the proposed framework utilizes the SGARCC, a hybrid architecture that combines graph-based modelling, recurrent mechanisms, and transformer-style attention. SGARCC represents log sequences as a graph $G = (V, E)$, where, V nodes correspond to individual log events and edges E represent temporal or semantic relationships between events [22].

The graph attention mechanism assigns learnable weights to each edge, enabling the model to focus on critical error propagation patterns while reducing the influence of noisy or irrelevant logs. The attention for a node i is computed as in equation (1):

$$\alpha_{ij} = \frac{exp\left(Leaky\ Re\ LU\left(a^T\left[Wh_i \| Wh_j\right]\right)\right)}{\sum_{k\in N(i)} exp\left(Leaky\ Re\ LU\left(a^T\left[Wh_i \| Wh_j\right]\right)\right)} \quad (1)$$

where, $W \in R^{d'} \times d$ is a learnable weight matrix, $a \in R^{2d'}$ is the attention vector, $N(i)$ is the neighbourhood of the node V_i, and $\|$ denotes vector concatenation. This graph-based attention ensures that the model focuses on highly informative error patterns, such as rare but critical failures, while attenuating less significant log events. Temporal correlations are captured by a recurrent structure, where the hidden state $h_t \in R^h$ at the time step t is updated according to equation (2):

$$h_t = \sigma\left(Ux_t + Vh_{t-1} + b\right) \quad (2)$$

where, $x_t \in R^d$ is the input feature vector, $U \in R^{h\times d}$ and $V \in R^{h\times h}$ are learnable weights, $U \in R^h$ is a bias term, and σ is a nonlinear activation function such as Rectified Linear Unit (ReLU). Figure 2 illustrates the architecture of the SGARCC model. By integrating graph attention with recurrent modelling, SGARCC captures both long-range temporal dependencies and inter-event relational dynamics, which are essential for modelling complex error propagation patterns in real-world log data. The latent representation z generated by SGARCC is mapped to one of the k error classes through a SoftMax function shown in equation (3):

$$P\left(y = c|z\right) = \frac{exp\left(w_c^T z + b_c\right)}{\sum_{k=1}^{K} exp\left(w_c^T z + b_k\right)} \quad (3)$$

where, w_c^T and b_c are learnable parameters class c. The predicted class $\hat{y}$ is defined as in equation (4):

$$\hat{y} = \arg\max_c P\left(y = c|z\right) \quad (4)$$

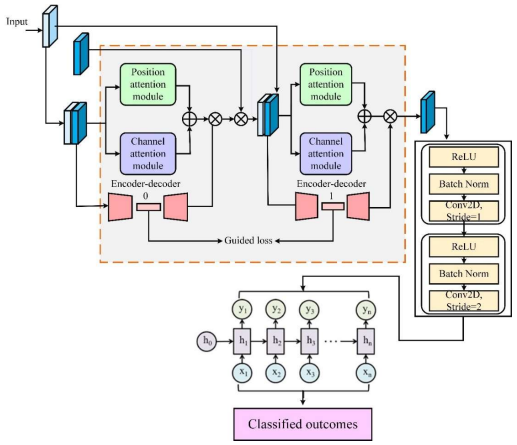


Figure 2: Architecture of SGARCC classification model

The model is trained using the cross-entropy loss, which is expressed as in equation (5):

$$L_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{K} y_{i,c}\ log\ P\left(y = c|z_i\right) \quad (5)$$

where, N is the number of log sequences, and $y_{i,c}$ is the one-hot encoded true label. The cross-entropy loss is chosen because it directly optimizes the probability distribution of the predicted classes relative to the ground truth, making it well-suited for multi-class classification problems, including the diverse runtime error types encountered in web applications. This loss ensures that the model assigns high confidence to the correct error category while penalizing misclassifications. To prevent overfitting and encourage smooth attention distributions, an L2 regularization term is applied to the attention weights shown in equation (6):

$$L_{reg} = \lambda\sum_{i,j}\alpha_{ij}^2 \quad (6)$$

where, λ is a regularization coefficient controlling the trade-off between classification accuracy and attention smoothness. The total loss function for training is thus illustrated in equation (7):

$$L_{total} = L_{CE} + L_{reg} \quad (7)$$

Minimizing L_{total} allows the model to simultaneously achieve high classification accuracy and maintain robust, interpretable attention weights, which are crucial for understanding why certain log sequences are flagged as anomalous. Once classified, the identified errors, including authentication errors, null errors, constraint violations, and endpoint errors, are passed to the LLM-based remediation module

[23][24]. The LLM generates semantically correct code snippets, which replace the original erroneous code, enabling the application to self-heal in real time. This integration of SGARCC-based detection with LLM-assisted correction ensures both robust anomaly detection and automatic error resolution, minimizing downtime and improving operational reliability.

2.3. Hyperparameter Optimization using Bighorn Sheep Optimization Algorithm

To enhance the robustness and performance of SGARCC, the BSOA is employed to optimize critical hyperparameters such as the learning rate η, dropout probability ρ_{drop}, and internal attention thresholds θ_{attn}. BSOA mimics the social and foraging behaviour of bighorn sheep, balancing exploration and exploitation to efficiently search the hyperparameter space [25]. The population of candidate solutions is initialized as in equation (8):

$$X_i^0 = X_{min} + r.(X_{max} - X_{min}), \quad i = 1,...,N \tag{8}$$

where, X_i^0 represents the ith candidate solution, X_{max} and X_{min} are the lower and upper bounds of the hyperparameters, and r is a random vector ensuring population diversity. Each candidate solution is evaluated using a fitness function that balances classification accuracy and inference latency, illustrated in equation (9):

$$F(X_i) = w_1.Accuracy(X_i) - w_2 Latency(X_i) \tag{9}$$

where, X_i, w_1 and w_2 are the weighting factors. Maximizing $F(X_i)$ ensures hyperparameters that achieve optimal real-time performance. Table 3 illustrates a comparative performance and functional analysis between the proposed BSOA and traditional gradient-based optimizers.

Table 3: **Comparative analysis of BSOA with traditional optimizers**

Criterions	SGD	Adam	RMSProp	Proposed BSOA
Optimization type	Gradient-based	Gradient-based	Gradient-based	Population-based metaheuristic
Search nature	Local	Semi-global	Semi-global	Global exploration–exploitation
Convergence speed	Moderate	Fast	Moderate	High with adaptive control
Escape from local minima	Poor	Moderate	Moderate	Excellent due to adaptive exploration
Parameter adaptivity	Fixed step size	Adaptive moment estimation	Adaptive learning rate	Dynamic control of learning rate, dropout, and attention thresholds
Sensitivity to initialization	High	Moderate	Moderate	Low, due to population diversity
Computational cost	Low	Moderate	Moderate	Moderate but highly parallelizable
Performance on noisy data	Poor	Moderate	Moderate	Robust due to global search and fitness regularization

Therefore, BSOA is integrated into the SGARCC framework to optimize key hyperparameters, including learning rate, dropout probability, and attention thresholds. Unlike gradient-based optimizers like SGD, Adam, or RMSProp, which rely on local gradients and risk local minima, BSOA employs a global population-based search. It initializes a diverse set of candidate solutions within defined bounds and evaluates them using a fitness function that balances classification accuracy and inference latency. Iterative exploration and exploitation enable efficient identification of optimal hyperparameters, enhancing SGARCC's convergence, robustness, and real-time performance for self-healing web applications.

2.4. Model Interpretability and Transition to Automated Correction

Once the SGARCC–BSOA pipeline successfully identifies and classifies runtime anomalies (such as

syntax errors, permission denials, and null pointer exceptions), the framework advances to its explainability and correction stage. This stage ensures that each decision made by the model is interpreted and validated before automated remediation is initiated. To achieve model interpretability, the proposed system incorporates the LIME technique [26]. LIME serves as a post-hoc explanation method, capable of interpreting any complex model by generating locally linear approximations around a specific prediction. The primary objective is to identify which features of the log data most significantly influenced the anomaly detection outcome, thereby ensuring transparency and trustworthiness in the decision-making process.

LIME approximates the original complex model $F(x)$ with an interpretable local surrogate model $h(z)$, where z denotes a locally perturbed representation of the original input x. The objective function that governs this approximation is expressed as in equation (10):

$$h^* = \underset{h \in H}{arg\,min}\, L(F, h, w_{x0}) + \Psi(h) \tag{10}$$

where, H represents the family of interpretable models, $L(F, h, w_{x0})$ denotes the local fidelity loss, measuring the deviation between the predictions of F and h in the vicinity of the target instance $x0$, w_{x0} is a locality kernel function that assigns higher importance to samples closer to $x0$, and $\Psi(h)$ is a regularization term controlling the complexity of the surrogate model to maintain interpretability. The locality weighting function is typically modeled as an exponential kernel, given by equation (11):

$$w_{x0}(z) = exp\left(- \frac{d(x_0, z)^2}{\tau^2} \right) \tag{11}$$

where, $d(x_0, z)$ is a distance function, and τ denotes the bandwidth parameter, controlling the influence radius around the instance $x0$. Through this process, LIME identifies the most influential input attributes (e.g., API call latency, HTTP response time, token parsing duration, user authentication frequency) that drive the anomaly classification outcome. The framework visualizes these feature contributions in a developer dashboard, enabling system operators to validate model decisions and trace the root causes of detected anomalies. After generating interpretable insights and validating the root causes of anomalies, the framework advances to an autonomous correction phase that intelligently formulates and applies precise code fixes without relying on predefined rules.

2.5. Automated Code Remediation Using Large Language Models

Following the anomaly detection and classification stage, the identified erroneous log sequences are forwarded to the LLM correction module, specifically Gemma3, for automated remediation. Let the set of detected errors be denoted as $\varepsilon = \{e_1, e_2, ..., e_m\}$, where each e_i corresponds to a classified error type from the SGARCC output [27].

For each error e_i, the corresponding code snippet C_i containing the anomaly is extracted from the application. The LLM takes as input a structured tuple (C_i, e_i, H_i), where, H_i represents the historical context, including past successful corrections, similar code patterns, and relevant log sequences. The LLM generates a corrected snippet $\hat{C}_i$ by maximizing the conditional probability shown in equation (12):

$$\hat{C}_i = \underset{C' \in S}{arg\,max}\, P(C' | C_i, e_i, H_i) \tag{12}$$

where, S is the space of syntactically valid Python code sequences, and $P(C' | C_i, e_i, H_i)$ is estimated by the LLM using its learned generative parameters. The LLM employs a transformer-based attention mechanism to capture both local syntactic dependencies and broader semantic context, ensuring that the generated code not only fixes the error but also preserves logical consistency with surrounding code. Once $\hat{C}_i$ is generated, it is evaluated against validation constraints to ensure correctness. Let $F(\hat{C}_i)$ denote a validation function that returns 1 if the snippet passes syntactic checks, runtime type checks, and context consistency, and 0 otherwise. The final code replacement is performed as in equation (13):

$$C_i^{new} = \begin{cases} \hat{C}_i & if\ F(\hat{C}_i) = 1 \\ C_i & Otherwise \end{cases} \tag{13}$$

This ensures that only verified corrections are applied to the live application, avoiding the introduction of new errors. The system iterates over all errors $e_i \in \varepsilon$ performing simultaneous correction while maintaining application stability. Let

$C^{updated} = \{C_1^{new}, ..., C_m^{new}\}$ represent the updated set of code snippets. The replacement process is formalized as a mapping function illustrated in equation (14):

$$R : C_i \rightarrow C_i^{new}, \forall i = 1, ..., m \quad (14)$$

where, R denotes the automated patching operator implemented by the LLM-based module. The final application code A^{fixed} is obtained by integrating all corrected snippets shown in equation (15):

$$A^{fixed} = A^{original} \setminus \{C_i\}_{i=1}^{m} \cup C^{updated} \quad (15)$$

This formalization ensures atomic and consistent code replacement, enabling real-time self-healing while preserving application functionality. The efficacy of the remediation process is further enhanced by feedback loops, where post-deployment logs are re-fed into SGARCC to monitor for residual or cascading errors. The combined framework of SGARCC-based detection and LLM-driven correction provides a theoretically grounded, fully automated mechanism for runtime anomaly detection and correction in Python web applications with minimal human intervention.

3. Experimental results

The experimental setup was implemented using Python 3.10 on an Ubuntu 22.04 server environment. The SGARCC framework was developed and executed within the FastAPI web framework, utilizing TensorFlow and PyTorch for deep learning operations. Model training and testing were performed on an NVIDIA RTX 4090 GPU with 24 GB VRAM, supported by CUDA 12.2. Log analysis, visualization, and interpretability were conducted using Pandas, Matplotlib, and LIME libraries.

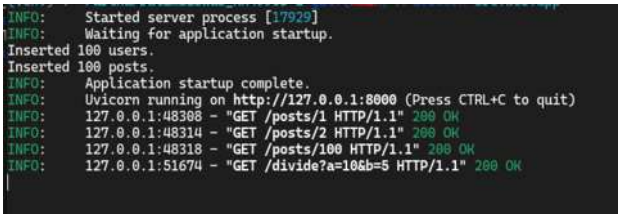
3.1. Hyperparameter settings

The hyperparameter configuration for the proposed SGARCC framework was optimized through systematic experimentation to ensure balanced convergence and stability. The learning rate was set to 0.001, optimized using the Adam optimizer with β_1 = 0.9 and β_2 = 0.999. The batch size was fixed at 64, with a total of 100 epochs for model training. The dropout rate of 0.3 was applied to prevent overfitting, while the hidden dimension in the recurrent compact block was set to 256. The Conviformer backbone employed 8 attention heads and 4 graph attention layers to enhance feature representation. Weight decay of 1e−5 was included for regularization, and the loss function was defined as categorical cross-entropy to ensure optimal multi-class anomaly classification.

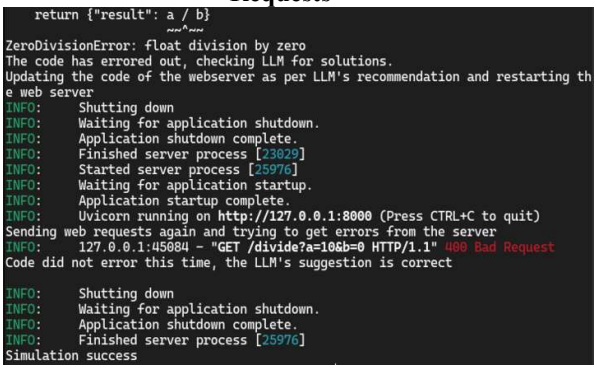
3.2. Implementation analysis

Figure 3 demonstrates a web server's ability to handle requests, detect errors, and perform autonomous self-healing. Figure 3(a) shows various curl commands sent to the server, such as retrieving posts and executing division operations, including cases where division by zero occurs, resulting in an internal server error. Figure 3(b) presents the server's initial response, logging each HTTP request and providing status codes like 200 OK for valid requests.

(a) Sending Web Requests to the Server



(b) Server Responding to Web Requests

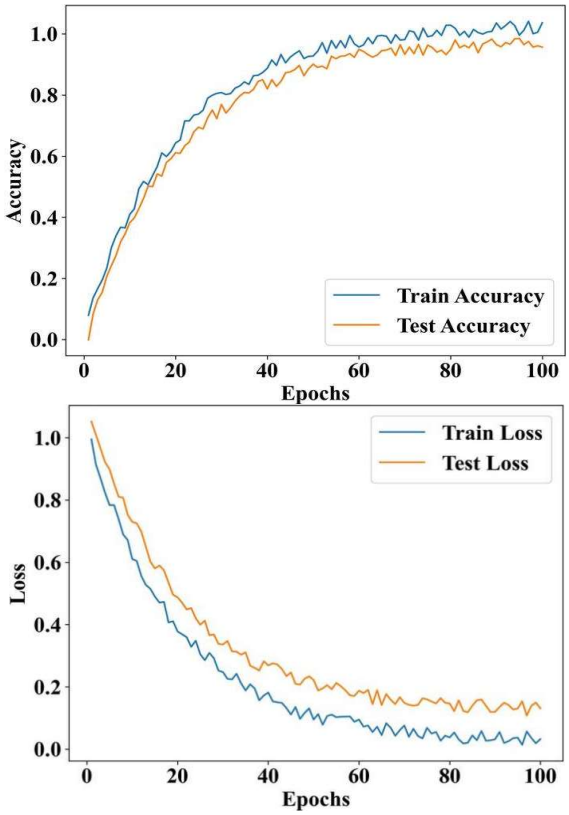


(c) Server Self-Healing After Errors and Resuming Operation

Figure:

Figure 3(c) illustrates the intelligent error recovery process when a ZeroDivisionError is encountered. The server uses an LLM to find solutions, updates its code, and restarts itself. It then verifies the fix by retrying the problematic request, which now correctly returns a 400 Bad Request instead of crashing. This sequence highlights the server's resilience, showing it autonomously detects runtime errors, seeks external

help, applies code fixes, and resumes normal operation without manual intervention.



(a)
(b)

Figure 4: Training and testing (a) accuracy and (b) loss analysis

Figure 4(a) shows that training and testing accuracy increase steadily with epochs, reaching about 0.99 and 0.97, respectively, after 100 epochs, indicating effective learning and minimal overfitting. Figure 4(b) depicts the corresponding loss curves, where training and testing losses drop consistently from around 1.0 to below 0.1, confirming convergence and stability of the proposed SGARCC framework for accurate and reliable runtime anomaly detection in web applications.

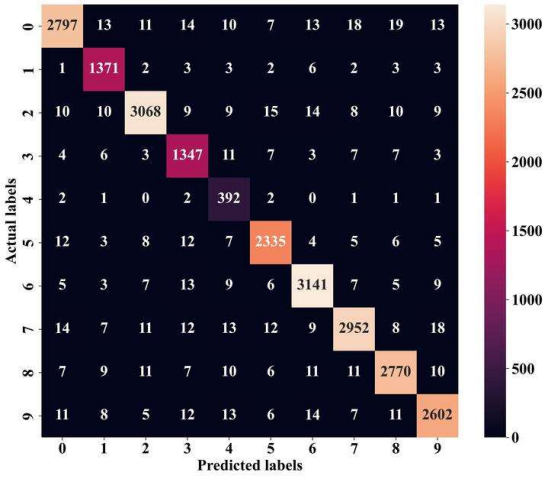
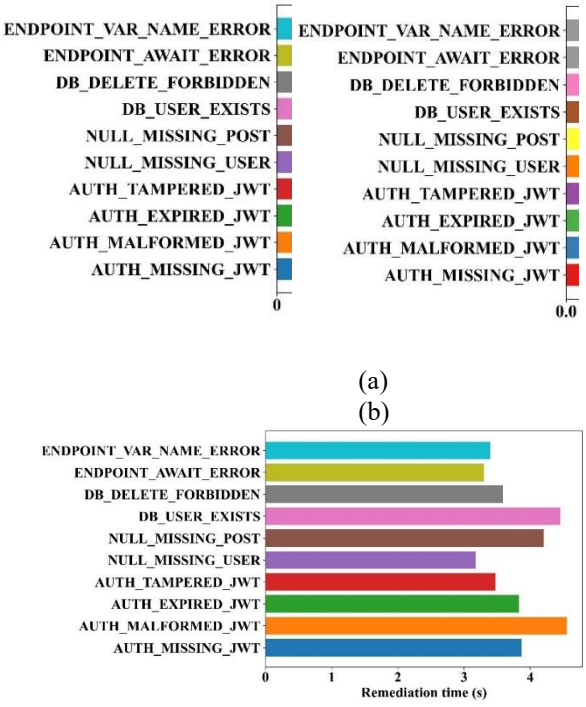


Figure 5: Confusion matrix

Figure 5 illustrates the confusion matrix, which illustrates the classification performance of the proposed SGARCC model, showing strong diagonal dominance with high true positive counts such as 2797, 3068, 3141, and 2952 for respective classes. Minimal off-diagonal errors indicate low misclassification. The model effectively distinguishes between different anomaly categories, achieving robust accuracy and consistency, confirming its reliability for runtime anomaly detection and classification in Python-based web applications.



(a)
(b)
(c)

Figure 6: Analysis of (a) Error distribution, (b) LLM-based inference time, and (c) Remediation time

Figure 6 presents a detailed quantitative evaluation of runtime anomalies detected and remediated by the proposed SGARCC + LLM framework in FastAPI-based web applications. The purpose of this analysis is to assess the system's detection frequency, inference efficiency, and remediation performance across multiple anomaly types. As shown in Figure 6(a), AUTH_MISSING_JWT and AUTH_EXPIRED_JWT were the most frequent anomalies (>3000 occurrences), while NULL_MISSING_USER occurred least (<800). Figure 6(b), the LLM inference time was highest for AUTH_MISSING_JWT (≈2.9 s) and lowest for ENDPOINT_AWAIT_ERROR (≈1.4 s). Figure 6(c), DB_USER_EXISTS required the longest remediation time (≈3.8 s), whereas AUTH_TAMPERED_JWT was corrected fastest (≈2.5 s), confirming the framework's robustness and real-time adaptability.

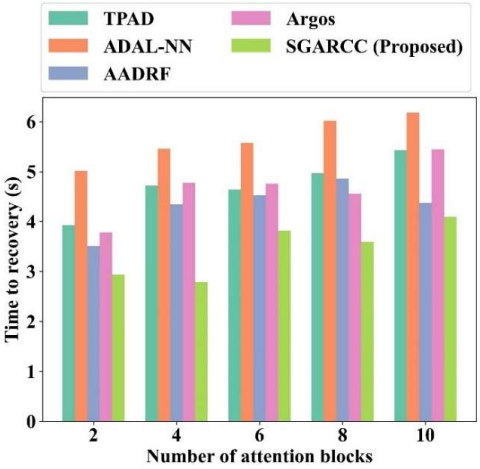


Figure 7: Analysis of recovery time variation with respect to the number of attention blocks for different anomaly detection models

Figure 7 illustrates the variation in recovery time with respect to the number of attention blocks for different anomaly detection models. The purpose is to assess how attention depth influences the framework's recovery efficiency. As the number of attention blocks increases from 2 to 10, the proposed SGARCC consistently exhibits the shortest recovery time, ranging from 3.0 s to 4.1 s, compared to TPAD (≈4.5–5.0 s), ADAL-NN (≈5.2–6.0 s), AADRF (≈4.2–4.8 s), and Argos (≈3.8–5.3 s), confirming SGARCC's superior scalability and responsiveness.

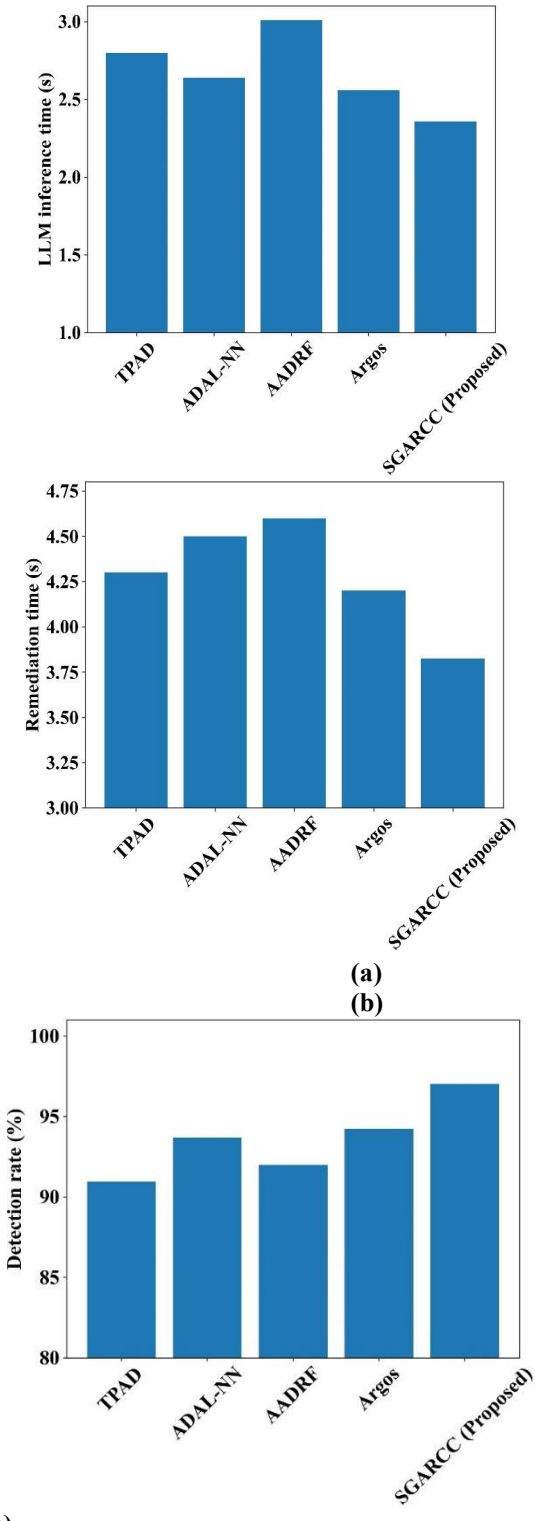


Figure 8: Comparison of (a) LLM inference time, (b) remediation time, and (c) detection rate

Figure 8 presents a comparative analysis highlighting the efficiency of the proposed SGARCC framework in runtime anomaly detection and remediation for web applications. The purpose of this evaluation is to

demonstrate how SGARCC enhances inference accuracy and response time compared to conventional methods such as TPAD, ADAL-NN, AADRF, and Argos. Figure 8(a), SGARCC achieved the lowest LLM inference time of 2.4 s, ensuring faster anomaly reasoning and response generation. Figure 8(b), the remediation time was the shortest at 3.7 s, validating the framework's rapid self-healing capability. Figure 8(c), SGARCC achieved the highest detection rate of 98%, confirming its superior precision in identifying anomalies while maintaining low latency, thereby optimizing both reliability and operational resilience.

Table 4: Comparison of Runtime Anomaly Detection Models

Models	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
RuntimeErrorSage [12]	90	88	87	87.5
FIRED [14]	91	89	88	88.5
TPAD [15]	92	90	91	90.5
ADAL-NN [16]	90	88	87	87.5
AADRF [17]	89	87	86	86.5
Argos [18]	93	91	92	91.5
SGARCC (Proposed)	**97.03**	**95.57**	**97.03**	**96.30**

Table 4 compares the performance of existing anomaly detection models against the proposed SGARCC framework. While traditional models like RuntimeErrorSage, FIRED, TPAD, ADAL-NN, AADRF, and Argos achieve accuracies between 89–93% with corresponding precision, recall, and F1-scores, SGARCC significantly outperforms them, achieving 97.03% accuracy, 95.57% precision, 97.03% recall, and a 96.30% F1-score. This demonstrates its superior ability to detect and classify runtime anomalies in web applications effectively.

Figure 8(a) presents the multi-class ROC curves for different runtime anomalies detected by the proposed SGARCC framework, demonstrating high classification performance with AUC values ranging from 0.98 to 0.99 across errors such as AUTH_MISSING_JWT, NULL_MISSING_USER, and DB_DELETE_FORBIDDEN.

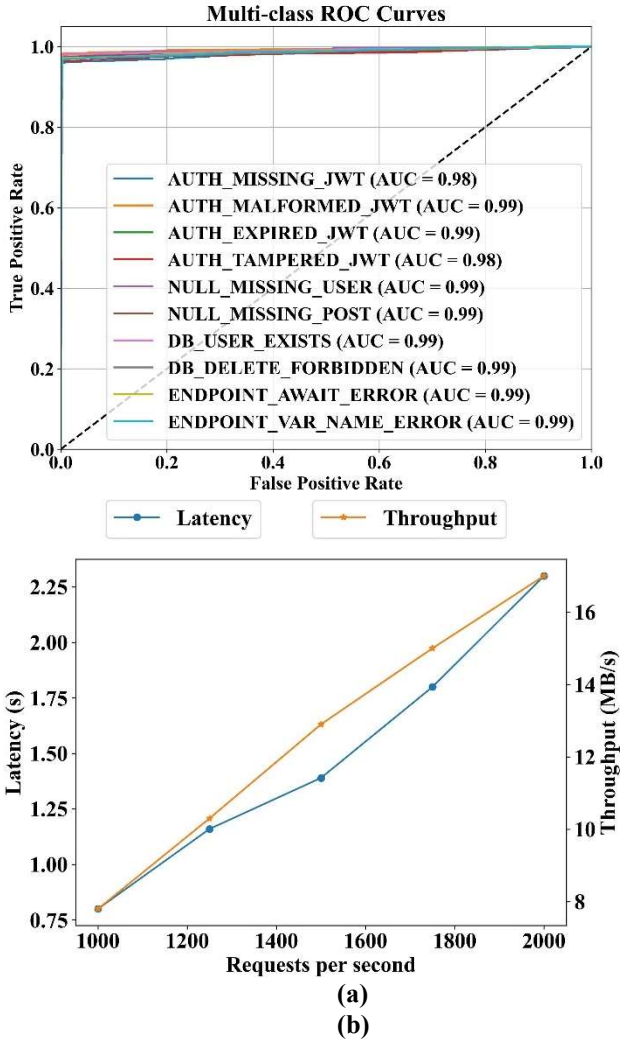
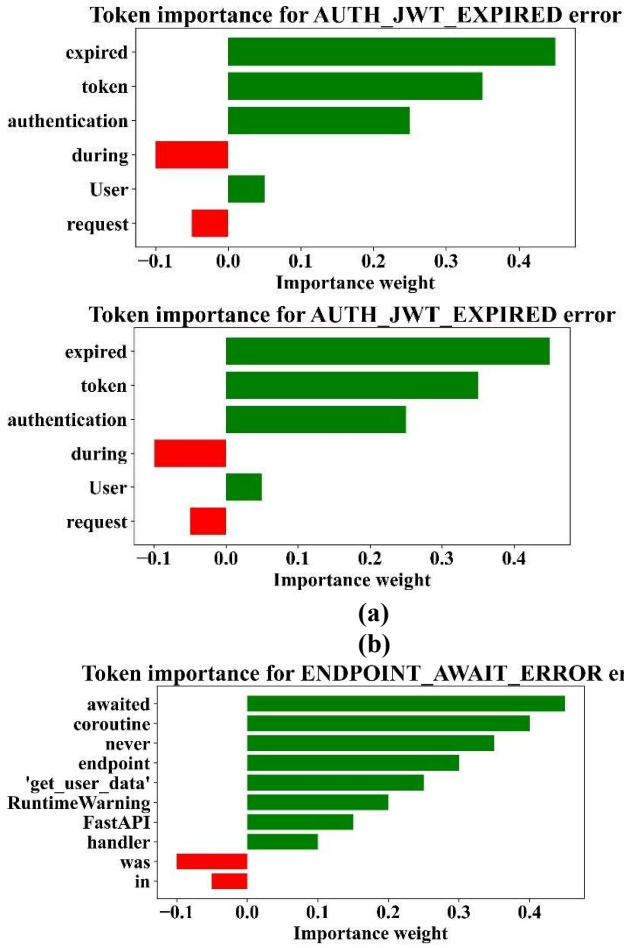


Figure 8: Analysis based on the error ROC curve and (b) latency and throughput of the proposed model

Figure 8(b) illustrates the model's efficiency, showing a nearly linear increase in latency and throughput as requests per second rise, indicating robust scalability. Together, these results confirm that the framework achieves precise anomaly detection while maintaining low-latency, high-throughput operation suitable for real-time Python web applications.



(c)

Figure 9: Interpretable analysis using LIME

Figure 9 illustrates the interpretable analysis of runtime anomalies using LIME within the proposed SGARCC-based framework. Figures 9(a) and 9(b) show token importance for the AUTH_JWT_EXPIRED error, highlighting that terms like "expired," "token," and "authentication" contribute most positively to detection, while less relevant tokens have a negative influence. Figure 9(c) presents token importance for the ENDPOINT_AWAIT_ERROR, where key tokens such as "awaited," "coroutine," and "endpoint" dominate prediction. These visualizations provide transparency into the model's decision-making by indicating which log features most strongly influence anomaly classification, enhancing interpretability and trust.

Table 5: Ablation study analysis of SGARCC model

Models	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Full SGARCC	**97.03**	**95.57**	**97.03**	**96.30**
w/o Graph Attention Layer	93.5	92.8	94.2	93.4
w/o Recurrent Compact Block	92.4	91.9	93.1	92.5
w/o Separate guidance mechanism	94.6	93.7	95.0	94.3
w/o Conviformer Backbone	91.8	90.9	92.6	91.7

Table 5 presents an ablation study illustrating the contribution of each core component within the proposed SGARCC model. The full SGARCC achieved the highest performance with 97.03% accuracy, 95.57% precision, 97.03% recall, and 96.30% F1-score, confirming its overall robustness. Removing the Graph Attention Layer reduced accuracy to 93.5%, while excluding the Recurrent Compact Block further dropped it to 92.4%. Similarly, omitting the Separate Guidance Mechanism and Conviformer Backbone resulted in 94.6% and 91.8% accuracy, respectively. These results highlight that all architectural components synergistically enhance the model's precision, recall, and stability for accurate anomaly detection and remediation.

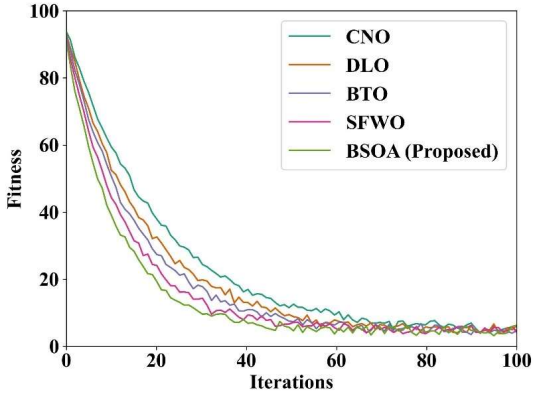


Figure 10: Fitness curve analysis

Figure 10 presents a comparison of five optimization algorithms, like Cellular Neighbours Optimizer (CNO) [28], Darco Lizard Optimizer (DLO) [29], Bermuda Triangle Optimizer (BTO) [30], Star Fish Optimization Algorithm (SFOA) [31], and BSOA (Proposed), by plotting their fitness values over 100 iterations. At the initial iteration (0), all algorithms start with a fitness value close to 100. The BSOA (Proposed) method consistently outperforms the

others, achieving lower fitness values earlier and converging faster. By iteration 60-80, all methods approach a fitness value close to 0, but the proposed BSOA maintains a slight edge throughout, demonstrating superior optimization capability and faster convergence.

3.3.Statistical Analysis and Evaluation

To validate the significance of the proposed SGARCC framework over baseline models, the Friedman test was conducted. Post-hoc pairwise comparisons using the Nemenyi test were performed to identify statistically significant differences between SGARCC and each baseline. All p-values below 0.05 indicate statistically significant superiority of SGARCC.

Table 6: Friedman test and post-hoc Nemenyi results showing significant performance differences (α = 0.05)

Models	Friedman Rank	p-value	Significant (α = 0.05)
SGARCC (proposed)	1.0	0.012	Yes
Transformer	2.5	0.038	Yes
GRU	3.0	0.027	Yes
LSTM	4.0	0.009	Yes

The performance of SGARCC and baseline models was evaluated using the Friedman test to determine whether statistically significant differences existed across multiple runs, followed by post-hoc Nemenyi analysis to identify pairwise distinctions. As shown in Table 6, SGARCC achieved the top Friedman rank of 1.0, while Transformer, GRU, and LSTM were ranked 2.5, 3.0, and 4.0, respectively. All models yielded p-values below 0.05, confirming that the observed differences are statistically significant at α = 0.05. These results demonstrate that SGARCC significantly outperforms the baseline models in runtime anomaly detection and automated remediation, highlighting its robustness and reliability in web application environments.

4. Discussions

The proposed SGARCC framework exhibited exceptional performance in runtime anomaly detection and automated remediation for web applications, achieving 97.03% accuracy, 95.57% precision, 97.03% recall, and a 96.30% F1-score. The model's robustness was further confirmed through ROC curve analysis, where AUC values ranged between 0.98 and 0.99 across anomaly classes such as AUTH_MISSING_JWT, DB_DELETE_FORBIDDEN, and NULL_MISSING_USER, indicating excellent classification consistency. The latency and throughput analysis demonstrated that the framework maintained near-linear scalability under high request loads, while the LLM inference time remained efficient, averaging around 2.4 seconds. Moreover, the framework's autonomous remediation mechanism achieved a minimum recovery time of 2.5 seconds for AUTH_TAMPERED_JWT and a maximum of 3.8 seconds for DB_USER_EXISTS, reflecting strong adaptability and real-time responsiveness. Statistical validation using the Friedman and Nemenyi tests ($p < 0.05$) confirmed the model's superiority and reliability compared to baseline architectures. Interpretability analysis using LIME further illustrated transparent decision-making by identifying dominant token contributions influencing anomaly classification.

- **Limitation**

A notable limitation lies in the model's dependency on LLM inference for code-level remediation, which introduces additional computational overhead during continuous high-load operations, suggesting the need for lightweight or on-device alternatives to optimize real-time self-healing efficiency in large-scale deployments.

4.1.Ethical implications

The proposed LLM-powered SGARCC framework introduces autonomous mechanisms for real-time anomaly detection and self-healing in web applications, which raises several ethical considerations. The integration of LLMs such as Gemma3 for automated code repair produces unintended or insecure patches if contextual understanding is incomplete, necessitating human-in-the-loop verification before deployment. Furthermore, since runtime monitoring and log collection may involve sensitive data, strict compliance with privacy and security standards such as GDPR and ISO/IEC 27001 is essential. Transparent audit trails are maintained to ensure accountability for all automatically generated code modifications. Ethical deployment of SGARCC, therefore, requires maintaining data privacy, explainability, fairness, and traceability to uphold responsible AI principles in automated remediation systems.

4.2.Threats to Validity

Although the proposed framework demonstrates superior performance across multiple metrics, several potential threats to validity should be acknowledged.

- **Internal Validity:** The experiments were conducted using runtime logs derived from controlled FastAPI-based web applications. These datasets fully represent real-world variability, where noise, network latency, or complex dependency chains influence anomaly detection accuracy.
- **External Validity:** The SGARCC architecture was specifically optimized for Python web applications. Its generalization to other programming environments remains to be

empirically validated. Differences in runtime semantics affect both detection precision and remediation latency.

- **Construct Validity:** Quantitative metrics to measure detection efficacy, but do not entirely capture the semantic validity of the LLM-generated corrections. Expert manual evaluation or code verification tools are required to assess the quality and correctness of proposed fixes.
- **Conclusion Validity:** The evaluation relied on a specific dataset and a fixed set of hyperparameters optimized via the BSOA. Although statistical performance improvements were significant, future replications using larger, heterogeneous datasets are necessary to confirm generalizability and mitigate potential overfitting biases.

5. Conclusion

This research presented a comprehensive runtime anomaly detection and self-healing framework for Python web applications built on FastAPI. The proposed framework continuously monitors and logs server data, generating rich, time-ordered insights into application behavior. At its core, the SGARCC effectively captures long-range temporal dependencies and relational patterns in log sequences through graph-based modeling combined with recurrent attention mechanisms. Hyperparameter tuning is dynamically performed using the BSOA**,** optimizing learning rates, dropout probabilities, and attention thresholds to enhance detection accuracy while minimizing inference latency. Detected anomalies are interpreted via LIME**,** providing feature-level transparency, and large language models such as Gemma3 generate context-aware, semantically correct code fixes by leveraging error logs, impacted code, and historical corrections. Future work may focus on extending the framework to support heterogeneous, multi-language microservice architectures and further reducing the computational overhead of LLM-based remediation. Incorporating predictive anomaly prevention and causal reasoning mechanisms could enable proactive error management, enhancing resilience and operational efficiency. Additionally, exploring lightweight or on-device models could improve real-time adaptability for large-scale deployments, ensuring scalable and robust self-healing capabilities across diverse web environments.

References

1. Reis MJ, Serôdio C. Edge AI for Real-Time Anomaly Detection in Smart Homes. Future Internet. 2025 Apr 18;17(4):179.
2. Kasri W, Himeur Y, Alkhazaleh HA, Tarapiah S, Atalla S, Mansoor W, Al-Ahmad H. From vulnerability to defense: The role of large language models in enhancing cybersecurity. Computation. 2025 Jan 29;13(2):30.
3. Ji X, Zhang L, Zhang W, Peng F, Mao Y, Liao X, Zhang K. LEMAD: LLM-Empowered Multi-Agent System for Anomaly Detection in Power Grid Services. Electronics. 2025 Jul 28;14(15):3008.
4. Osa E, Orukpe PE, Iruansi U. Design and implementation of a deep neural network approach for intrusion detection systems. e-Prime-Advances in Electrical Engineering, Electronics and Energy. 2024 Mar 1;7:100434.
5. Cen M, Deng X, Jiang F, Doss R. Zero-Ran Sniff: A zero-day ransomware early detection method based on zero-shot learning. Computers & Security. 2024 Jul 1;142:103849.
6. Nazat S, Li L, Abdallah M. XAI-ADS: An explainable artificial intelligence framework for enhancing anomaly detection in autonomous driving systems. Ieee Access. 2024 Apr 2;12:48583-607.
7. Maddu NM, Rao YN. Integrated Intrusion Detection and Mitigation Framework for SDN-Based IIOT networks using lightweight and adaptive AI techniques. Journal of Information Systems Engineering & Management. 2025;10(9s):456-72.
8. James UU, Idika CN, Enyejo LA, Abiodun K, Enyejo JO. Adversarial Attack Detection Using Explainable AI and Generative Models in Real-Time Financial Fraud Monitoring Systems. International Journal of Scientific Research and Modern Technology. 2024 Dec 29;3(12):142-57.
9. Partha Pratim Biswas. (2024). Controlling Runtime-Anomaly of A Web Application Using Advanced Machine Learning. International Journal of Intelligent Systems and Applications in Engineering, 12(20s), 1006 –. Retrieved from https://ijisae.org/index.php/IJISAE/article/view/7216
10. Xu W, Luo J, Huang T, Sui K, Geng J, Ma Q, Akasaka I, Shi X, Tang J, Cai P. A Two-Staged LLM-Based Frameswork for CI/CD Failure Detection and Remediation with Industrial Validation. arXiv preprint arXiv:2506.03691. 2025 Jun 4.
11. Toprani D, Madisetti VK. LLM Agentic Workflow for Automated Vulnerability Detection and Remediation in Infrastructure-as-Code. IEEE Access. 2025 Apr 15.
12. Yonathan M. RuntimeErrorSage: Intelligent Runtime Error Analysis and Remediation using Local Large Language Models. Available at SSRN 5367078. 2025 May 26.
13. Alhanahnah M, Rashedul Hasan M, Xu L, Bagheri H. An empirical evaluation of pre-trained

large language models for repairing declarative formal specifications. Empirical Software Engineering. 2025 Sep;30(5):1-38.
14. Xin R, Chen P, Grosso P, Zhao Z. A fine-grained robust performance diagnosis framework for run-time cloud applications. Future Generation Computer Systems. 2024 Jun 1;155:300-11.
15. Ma S, Guan S, He Z, Nie J, Gao M. TPAD: Temporal-pattern-based neural network model for anomaly detection in multivariate time series. IEEE Sensors Journal. 2023 Nov 6;23(24):30668-82.
16. Otieno C. AI-Enhanced Anomaly Detection and Autonomous Remediation Framework for Securing UEFI BIOS Firmware in Heterogeneous Computing Platforms. Available at SSRN 5390668. 2025 Aug 8.
17. Ahmed K, Altaf A, Jamail NS, Iqbal F, Latif R. ADAL-NN: Anomaly detection and localization using deep relational learning in distributed systems. Applied Sciences. 2023 Jun 19;13(12):7297.
18. Gu Y, Xiong Y, Mace J, Jiang Y, Hu Y, Kasikci B, Cheng P. Argos: Agentic time-series anomaly detection with autonomous rule generation via large language models. arXiv preprint arXiv:2501.14170. 2025 Jan 24.
19. Tran E. A Hybrid Generative AI–Driven Automation Model for Intelligent Vulnerability Detection and Remediation in Cloud-Based Machine Learning Systems. Journal Code. 2024;1443:7126.
20. Luis D, Talha R, Oladele S, Racheal S. Development of an Intelligent Server Monitoring System using Machine Learning Algorithms for Real-Time Anomaly Detection and Automated Response.
21. Luo X, Jha SM, Sinha A, Li Z, Liu Y. ALPHA: LLM-Enabled Active Learning for Human-Free Network Anomaly Detection. arXiv preprint arXiv:2509.05936. 2025 Sep 7.
22. Zhu P, Wang B, Tang K, Zhang H, Cui X, Wang Z. A knowledge-guided graph attention network for emotion-cause pair extraction. Knowledge-Based Systems. 2024 Feb 28;286:111342.
23. Manalu HV, Rifai AP. Detection of human emotions through facial expressions using hybrid convolutional neural network-recurrent neural network algorithm. Intelligent Systems with Applications. 2024 Mar 1;21:200339.
24. Vaishnav M, Fel T, Rodríguez IF, Serre T. Conviformers: Convolutionally guided vision transformer. arXiv preprint arXiv:2208.08900. 2022 Aug 17.
25. Wang X. Bighorn Sheep Optimization Algorithm: a novel and efficient approach for Wireless Sensor Network coverage optimization. Physica Scripta. 2025 Jun 10.
26. Zhang Y, Chen L, Tian Y. A Method for Evaluating the Interpretability of Machine Learning Models in Predicting Bond Default Risk Based on LIME and SHAP. arXiv preprint arXiv:2502.19615. 2025 Feb 26.
27. Vo NP, Paulovicks B, Sheinin V. LLM-as-a-Judge for Reference-less Automatic Code Validation and Refinement for Natural Language to Bash in IT Automation. arXiv preprint arXiv:2506.11237. 2025 Jun 12.
28. Barba-Toscano O, Cuevas E, Escobar-Cuevas H, Islas-Toski M. Cellular neighbors optimizer: a novel metaheuristic approach inspired by the cellular automata and agent-based modeling for global optimization. The Journal of Supercomputing. 2025 May 21;81(8):869.
29. Wang X. Draco lizard optimizer: a novel metaheuristic algorithm for global optimization problems. Evolutionary Intelligence. 2025 Feb;18(1):10.
30. Shehadeh HA. Bermuda Triangle Optimizer (BTO): A Novel Metaheuristic Method for Global Optimization. International Journal of Advances in Soft Computing and its Applications. 2025 Jul;17(2).