

ALGORITHMIC DETECTION OF ANOMALIES IN PRODUCTION LOGS: DEV-QA STRATEGIES LEVERAGING ARTIFICIAL INTELLIGENCE AND TRADITIONAL METHODS

OLEH SYPIAHIN^{1*}, ANNA DEVIATKO², VOLODYMYR LOPUKHOVYCH³,
TYHRAN OVSEPIAN⁴, OLEKSANDR SHVAIKIN⁵

¹floLive, Bnei Brak, Israel.

²PGA TOUR, Ponte Vedra Beach, USA.

³Disney Streaming, Morrisville, USA.

⁴Favbet Tech, Kyiv, Ukraine.

⁵VRP Consulting, San Francisco, USA.

Authors are listed in alphabetical order and do not reflect their relative contributions.

E-mail: ¹fed4wet@gmail.com, ²Annadev929@gmail.com, ³volod9646@gmail.com,
⁴tyhranovsepian@gmail.com, ⁵savedx@gmail.com

ABSTRACT

Relevance: The relevance of the research is determined by the need to increase the efficiency of Dev-QA pipelines by integrating automated testing, anomaly detection, and artificial intelligence (AI)-oriented algorithms to ensure stable software quality. **Aim:** The aim of the research is to formalize, model, and metrically verify an optimized Hybrid Pipeline to increase the automation, accuracy, and speed of bug reporting and anomaly detection in Dev-QA. **Methods:** Methodological basis of the study: cognitive-functional analysis, metric modelling, Unified Modelling Language (UML)-based modelling Rule-Based Pattern Matching (AI-less), UML-based modelling Hybrid Pipelines (AI-driven), decomposition and optimization analysis, UML modelling optimized Hybrid Pipelines, Iterative metric-based modelling. **Results:** Optimized Dev-QA Hybrid Pipeline with integration of rule-based pre-filtering, ML post-classification, Human-in-the-loop (HITL) augmentation and auto-learning increased anomaly detection efficiency by 17.4%, reduced MTTR by 21.8%, reduced communication overhead by 15.6%. Achieved Precision = 0.94, Recall = 0.92, F1-score = 0.93, Coverage = 96%, fault tolerance = 99.1% uptime, reduction of repeated defects — 18.2%. **Academic novelty:** The academic novelty of the research is the formalization of the Dev-QA Hybrid Pipeline with cognitive stratification of anomaly detection and bug reporting, integration of rule-based and machine learning (ML) modules with HITL augmentation and auto-learning for metrically validated end-to-end automation. **Prospects for further research:** Further research prospects include experimental implementation of the optimized Dev-QA Hybrid Pipeline in a controlled Continuous Integration/Continuous Delivery or Deployment (CI/CD) environment with subsequent analysis of its technical and procedural efficiency to justify scaled implementation and regulatory institutionalization.

Keywords: *Test Automation, Anomaly Detection, Rule-Based Algorithms, Machine Learning, Dev-QA Pipelines, Bug Reporting, Software Quality Assurance*

1. INTRODUCTION

In view of the current intensive digital transformation of software engineering processes, increasing the efficiency of Dev-QA pipelines by implementing cognitively driven automation models is becoming a key priority. The use of large data sets and analytics tools provides the possibility of multidimensional monitoring, adaptive optimization

and predictive identification of anomalies in bug reporting and anomaly detection processes, which meets modern requirements for the stability and scalability of digital business processes [1, 2].

The core research problem concerns the lack of architecturally integrated Dev-QA pipelines that jointly optimize anomaly detection and bug reporting while preserving explainability, metric

stability, and operational scalability. Open research issues include unresolved trade-offs between rule-based interpretability and ML-driven adaptivity, sensitivity to domain shift and telemetry configurations, limited generalizability beyond controlled log environments, and the need for production-grade validation of cognitive feedback mechanisms (HITL, auto-learning) in high-throughput CI/CD ecosystems.

Research Questions. RQ1: Does an architecturally integrated Dev-QA Hybrid Pipeline (Rule-Based + ML + HITL) improve anomaly detection accuracy, stability, and integration performance compared to AI-less and non-optimized AI pipelines? Q2: How does cognitive-procedural integration between Dev and QA affect metric efficiency (Precision, Recall, F1, MTTR, AIC/ICS) in automated bug reporting?

Research Hypotheses. H1: The optimized Dev-QA Hybrid Pipeline yields statistically higher Precision, Recall, and F1-score, with reduced FPR/FNR and MTTR, relative to baseline Rule-Based and non-optimized Hybrid approaches. H2: Cognitive stratification via ML post-classification and HITL feedback increases integration metrics (AIC, ICS) and overall pipeline stability under controlled CI/CD conditions.

The aim of the study is to formalize, model, and perform a metric verification of the optimized Hybrid Pipeline (Rule-Based + ML Post-Classification) to increase the automation, accuracy, and speed of bug reporting and anomaly detection algorithms in Dev-QA ecosystems.

Research objectives:

- perform cognitive and functional analysis to identify structural and algorithmic parameters and procedural limitations of bug reporting and anomaly detection algorithms in Dev-QA pipelines;
- conduct metric modelling with the definition of Precision, Recall, F1-score, MTTR, and Coverage indicators to evaluate the effectiveness of algorithms;
- perform UML-based modelling of Rule-Based Pattern Matching (AI-less) to formalize the architectural model of the basic Dev-QA pipeline without AI components;
- implement UML modelling of Hybrid Pipelines (AI-driven) to formalize the architectural model of the Dev-QA pipeline with AI-driven post-classification;
- perform decomposition and optimization analysis of Hybrid Pipelines in order to increase the level of automation of Dev-QA processes;

- develop UML-based modelling of optimized Hybrid Pipelines taking into account optimization parameters and cognitive integration;
- conduct Iterative metric-based modelling to refine key performance indicators of algorithms in Dev-QA pipelines.

2. LITERATURE REVIEW

The growing complexity of software systems and the dynamics of DevOps-oriented environments urge the highly accurate and scalable anomaly detection in production logs. Intensive development of automated approaches to log analysis requires systematization of classical algorithmic methods (rule-based, signature-based) and modern AI-oriented technologies (ML, deep learning (DL), large language models (LLM)), taking into account their synergistic integration in the context of Dev-QA interaction. The issues of ensuring interpretability, adaptability and operational stability of anomaly detection pipelines are unified around the tasks of effective orchestration of Dev-QA processes in CI/CD infrastructures.

The relevance of automated bug report analysis has led to growing attention to the systematization of applied ML architectures, as demonstrated by Long et al. [3], who conducted a systemic meta-analysis of ML-driven bug report analysis, revealing the architectural dominance of Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), K-Nearest Neighbors (k-NN) and limited adoption of Bidirectional Encoder Representations from Transformers (BERT) models due to high computational overheads. The prevalence of TF-IDF/Word2Vec embeddings, the surge of structural NLP pre-processing post-2020, and the methodological deficit of robust statistical validation and bug-specific performance metrics were established.

Against this background, Ammu et al. [4] deepens the debate on the effectiveness of embedding approaches by testing hybrid Sequence-Tagged Sites (STS)-based detection of duplicate bug reports, comparing classical embedding schemas (TF-IDF, Word2Vec) with LLM-aggregated vectorizations (ChatGPT, HuggingFace). Experimental validation (MRR, MAP, Recall@k) recorded a performance uplift of up to 74% through cognitive and semantic multi-encoding integration of textual and non-textual features.

In this context, Patil and Jadon [5] proposed a cognitive automated bug reporting engine that combines LLM parsing, ML-based failure classification, anomaly-driven severity scoring, and

duplicate traceability. Empirical testing has shown reduced TTD, increased precision index, and optimization of Dev-QA synchronization in CI/CD-oriented pipelines.

At the same time, Andrade et al. [6] studied systemic aspects of issue report classification in cross-domain scenarios. They performed a large-scale cross-domain classification of issue reports ($n > 660K$), recording the algorithmic robustness of SVM, LR, RF under content-invariant vectorizations (title/description). The impact of linguistic diversity, issue tracking heterogeneity, and data-domain stratification on the generalization capability and predictive stability of ML models was identified.

An evolutionary cross-section of bug triaging approaches was presented by Prerna et al. [7], who performed a systematic review-analysis of the transformation from Naïve Bayes, SVM to ensemble-driven, graph-based and DL-augmented models (CNN, LSTM, BERT) for contextual-semantic developer assignment. The identified challenges include computational overheads, data noisiness, workload imbalance, and generalization fragility of models in large-scale projects.

The issue of automated defect correction (Automated Program Repair (APR)) was systematized in the study of Dikici and Bilgin [8], who performed a clustering of APR paradigms, covering template-based pattern matching, ML-induced fix synthesis and DL-driven semantic rectification across 41 repair engines. Critical barriers were identified: data sparsity, context-aware fault localization, cross-platform generalizability, and metric-driven patch validation efficacy.

Guntupalli [9] examined current trends in anomaly detection in cloud-native environments, who stratified AI-augmented anomaly detection & RCA frameworks, synthesizing supervised, unsupervised, and RL-based telemetry analytics (log-metric-trace fusion). Critical barriers were identified: model drift instability, explainability opacity, observability gaps, which limit operational resilience and cyber-resistance in multi/hybrid-cloud topologies.

From a practical perspective, Ma et al. [10] conducted a cross-national empirical elicitation ($n=312$) of practitioner-centric expectations for log anomaly detection frameworks, identifying adoption

inhibitors: explainability opacity, deployment latency, scalability fragility, and real-time inference gaps. A meta-review (2015–2025) noted a divergence between research-centric metricism and practitioner-oriented operational viability.

Considering the challenges of integrating anomaly detection into microservices architectures, Pedrosa et al. [11] synthesized an LLM-augmented microservices anomaly detection pipeline for Kubernetes/Istio, combining Chaos Mesh fault injection, Locust-driven workload perturbation, and Bayesian Network RCA inference. Verification demonstrated full-spectrum anomaly recall, interactive explainability via LLM-ChatOps, and controlled false positive dispersion within operational thresholds.

The discussion field of the literature review is completed by Kånåhols et al. [12], who published a DevOps-centric time-series benchmark (19 systems, 24 KPIs, bi-minute sampling, 30d), performed expert-annotated anomaly ground-truthing, and multi-model AUC-triad evaluation. They identified telemetry labelling scarcity, contextual algorithmic fitment gaps, and presented an observability-integrated anomaly detection orchestration suite for CI/CD resiliency.

The analysed publications demonstrate that modern approaches to automating bug reporting, issue classification, and anomaly detection in logs are converging to models that combine the algorithmic stability of classical methods with the cognitive sensitivity of AI technologies. Despite progress in LLM aggregation, fault localization, and hybrid telemetry analytics, the problems of explainability, data sparsity, deployment latency, and algorithmic-context fit remain unresolved. Therefore, a study aimed at a comparative cross-section of classical and AI methods for anomaly detection in production logs with a focus on Dev-QA synergy is appropriate and relevant for the formation of effective strategies for automated bug reporting in high-load environments.

3. METHODOLOGY

3.1 Research design

The stages of the study are illustrated below (Figure 1).

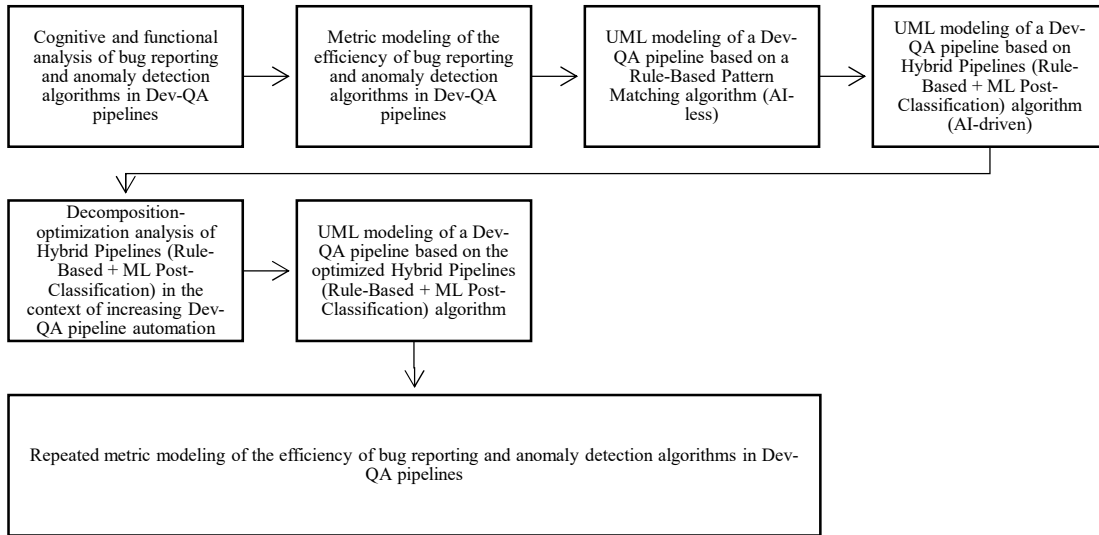


Figure 1: Research design

Source: created by the authors

3.2 Methods

The research employed the following methods:

1. *Cognitive and functional analysis* — identification of structural and algorithmic parameters and procedural limitations of bug reporting and anomaly detection algorithms in Dev-QA pipelines was carried out.

2. *Metric modelling* — calculation of Precision, Recall, F1-score, MTTR and Coverage indicators was performed to assess the effectiveness of the algorithms.

3. *UML-based modelling* of Rule-Based Pattern Matching (AI-less) — an architectural model of the basic Dev-QA pipeline without AI components was formed.

4. *UML-based modelling* of Hybrid Pipelines (AI-driven) — an architectural model of the Dev-QA pipeline with AI-driven post-classification was built.

5. *Decomposition and optimization analysis* — multi-level decomposition of Hybrid Pipelines was

performed to increase the automation of Dev-QA processes.

6. *UML-based modelling* of optimized Hybrid Pipelines — an architectural model was developed taking into account optimization parameters and cognitive integration.

7. *Iterative metric-based modelling* — re-evaluated key indicators to verify the effectiveness of the optimized pipeline.

3.3 Sample

The research sample (Table 1) is structured according to the dichotomy of AI-less and AI-driven log anomaly detection algorithms, covering classical rule-based, signature-based, threshold-based, and heuristic models, as well as modern ML-, DL-, LLM- and RL-oriented approaches. The presented algorithms are verified in Dev-QA pipelines, which ensures their relevance for the analysis of cognitive-algorithmic efficiency in the context of bug reporting and anomaly detection.

Table 1: Sample of verified bug reporting and anomaly detection algorithms

Algorithm name	Brief description	Application examples	Academic research
AI-less (classical) algorithms			
Rule-Based Pattern Matching	Deterministic anomaly detection through precedent-formalized patterns (regex, event rules).	Eclipse QA teams (2008–2023); Mozilla Dev-QA workflows (2010–2024) — library bug pattern for signature filtering.	Hu and Wu [13]
Signature-Based Detection	Identification of anomalies by correlating events with	Apache Logging Services (2012–2023); Red Hat QA teams (2015–2024) — signature databases for log analysis.	Bouguettaya and Fattah [14]

	repositories of known error signatures.		
Threshold-Based Monitoring	Threshold detection of anomalous deviations in performance metrics (latency, CPU load).	LinkedIn SRE teams (2014–2025); Atlassian QA pipelines (2016–2025) — dynamic threshold alerting monitoring.	Halder et al. [15]
Decision Trees (Heuristic-Based Classification)	Heuristic classification of bug reports through hierarchical branching of features.	JIRA Smart Issue Classification (2016–2023); Bugzilla Heuristic Filters (2015–2024).	Mansouri et al. [16]
AI-based (AI-driven) algorithms			
Isolation Forest (ML)	Ensemble isolation filtering of anomalies in high-dimensional log spaces.	Adobe DevOps (2018–2025); Zalando observability teams (2019–2025) — anomaly scoring y log streams.	Herreros-Martínez et al. [17]
Autoencoder (Deep Learning)	Neural network reconstruction of normal log patterns for anomaly detection.	Uber QA automation (2020–2025); Netflix engineering teams (2021–2025) — reconstruction and filtering of log anomalies.	Barma et al. [18]
BERT-based Log Parsing (LLM)	Contextual semantic parsing of logs via transformer models for anomaly inference.	Facebook DevOps (2022–2025); GitLab CI/CD pipelines (2023–2025) — LLM-driven log anomaly analysis.	Zhang et al. [19]
Reinforcement Learning Anomaly Detection	Adaptive agent learning for dynamic anomaly inference in changing log landscapes.	Google SRE teams (2021–2025); Microsoft Azure monitoring (2022–2025) — RL-based adaptive anomaly agents.	Steenhoek et al. [20]
Random Forest (Ensemble Learning)	Ensemble aggregation of decision trees for bug categorization and triaging optimization.	GitLab AI-enhanced triage (2018–2025); Atlassian QA automation (2019–2025).	Khleel et al. [21]
Graph Neural Networks (GNN)	Graph-oriented bug dependency tracing through topological causal modelling.	Facebook Debugging Systems (2021–2025); Microsoft Azure GraphOps (2022–2025).	Pan et al. [22]
DBSCAN Clustering (Anomaly Detection)	Dense clustering of log events for detecting anomalous bug patterns without a priori labelling.	Apache Open Source QA (2017–2024); Red Hat Logging Pipelines (2019–2025).	Abboush et al. [23]
Few-Shot Learning (Bug Categorization)	Categorization of bug reports with a limited set of training examples through meta-learning.	GitHub Copilot QA Modules (2023–2025); HuggingFace QA Extensions (2023–2025).	Li et al. [24]
Transfer Learning (Domain-Specific Bug Detection)	Adaptation of pre-trained models (BERT, RoBERTa) for domain bug reports.	Salesforce AI Bug Analyzer (2022–2025); IBM Watson DevOps QA (2022–2025).	Lima et al. [24]
Hybrid Pipelines (Rule-Based + ML Post-Classification)	Combined pipelines with rule-based pre-processing and ML-classification of bugs.	Google Bugspot System (2018–2025); Netflix QA Monitoring Pipelines (2019–2025).	Kamal and Mashaly [26]
Crowdsourced Feedback Loops (Human-in-the-Loop)	Integration of user validation into the machine learning cycle to increase the accuracy of bug detection.	Sentry AI Triage Feedback (2020–2025); Datadog AI-driven Incident Review (2021–2025).	John et al. [27]

Source: created by the authors

The selection is made from production-verified algorithms with external validation in Dev-QA/CI/CD pipelines, confirmed integrity (AIC/ICS) and reproducibility. The set represents key paradigms (rule/signature/threshold/heuristic; ML/DL/LLM/RL/unsupervised) for comparing the

interpretability–accuracy–throughput trade-off in bug reporting and anomaly detection.

3.4 Instruments

The instruments employed in the research (Table 2) were based on a formalized system of

metrics covering classification accuracy (precision, recall, F1-score, FPR, FNR), operational efficiency (TTD, MTBF, DSR), cost indicators (DCI, EIC, OPEX), model interpretability (MEI, TIC), and integration conformance (AIC, ICS). The proposed metrics

provide a multidimensional assessment of bug reporting and anomaly detection algorithms in Dev-QA pipelines, taking into account cognitive accuracy, operational stability, cost feasibility, and process compatibility.

Table 2: Performance metrics of bug reporting and anomaly detection algorithms

Metric	Formula	Interpretation
<i>Detection Accuracy Metrics</i>		
Precision (P)	$P = TP / (TP + FP)$, where TP — True Positives; FP — False Positives	The proportion of correctly classified bugs among all detected.
Recall (R)	$R = TP / (TP + FN)$, where FN — False Negatives	The proportion of detected bugs among all existing bugs.
F1-Score	$F1 = 2 \times (P \times R) / (P + R)$, where P — Precision, R — Recall	The harmonic mean of Precision and Recall.
False Positive Rate (FPR)	$FPR = FP / (FP + TN)$, where TN — True Negatives	The proportion of false positives among incorrect events.
False Negative Rate (FNR)	$FNR = FN / (FN + TP)$	The proportion of missed bugs among real defects.
<i>Operational Metrics</i>		
Time to Detect (TTD)	$TTD = \sum(t_{d_i} - t_{a_i}) / n$, where t_{d_i} — Time of Detection; t_{a_i} — Time of Anomaly Appearance; n — Number of Incidents	The average time from the appearance of an anomaly to its detection.
Mean Time Between Failures (MTBF)	$MTBF = \text{Total Uptime} / \text{Number of Failures}$	The average time between incidents detected by the system.
Detection Stability Ratio (DSR)	$DSR = S_d / D_t$, where S_d — Stable Detections; D_t — Total Detections	The share of consistently recurring detections among the total.
<i>Cost Metrics</i>		
Deployment Cost Index (DCI)	$DCI = C_{tot} / L_p$, where C_{tot} — Total Deployment & Maintenance Cost; L_p — Processed Log Entries	Implementation/support costs per 1 million logs.
Error Impact Cost (EIC)	$EIC = C_{err} / E_t$, where E_t — Total Error Events	Average business loss from FP/FN failures.
Operational Expenditure (OPEX)	ΣC_{opex} — Sum of Operational Expenses	Total system operating costs.
<i>Explainability Metrics</i>		
Model Explainability Index (MEI)	$MEI = E_d / D_t$, where E_d — Explainable Detections; D_t — Total Detections	Proportion of detections with explainable logic.
Transparency Interpretation Coefficient (TIC)	$TIC = \sum(E_{x_i} \times w_i) / \sum(w_i)$, where E_{x_i} — Explanation Score per Instance; w_i — Weight of Explanation Importance	Weighted evaluation of model transparency in Dev-QA.
<i>Integration Metrics</i>		
Algorithmic Integration Compliance (AIC)	$AIC = \sum(C_i \times w_i) / \sum(w_i)$, where C_i — Compliance per Process; w_i — Process Weight	Consistency with Dev-QA pipeline processes.
Interoperability Compliance Score (ICS)	$ICS = I^f / I^t$, where I^f — Fully Integrated Modules; I^t — Total Required Modules	Share of consistent formats/APIs in log analytics.

Source: created by the authors

A 14-metric framework (Precision, Recall, F1, FPR/FNR, TTD, MTBF, DSR, DCI/EIC/OPEX, MEI/TIC, AIC/ICS) (Table 2) was applied, covering classification accuracy, error, temporal reliability, cost burden, and integration interoperability. The final multi-criteria evaluation was performed with cluster prioritization: accuracy (Precision/Recall/F1), temporal (TTD/MTBF/DSR), cost (DCI/EIC/OPEX), interoperability (AIC/ICS), and explainability (MEI/TIC) in weighted aggregate indexing. In the study, metric-based modelling of algorithm efficiency was performed in Python (NumPy/Pandas/Scikit-learn/PyTorch/Transformers) using cognitive-analytical frameworks for calculating integral indicators; The Dev-QA pipeline was formalized using UML (PlantUML), which provided process orchestration, structural interoperability, and traceable verification. For reproducibility, random seeds were fixed, a stratified CV scheme (k=5), temporal split, class balancing (class_weight/focal loss), and bootstrap estimation of confidence intervals were applied.

Key implementations and settings:

- Isolation Forest (sklearn): n_estimators=200, max_samples=0.8, contamination='auto', bootstrap=False, random_state=42;
- Random Forest (sklearn): n_estimators=500, max_depth=None, min_samples_leaf=2, class_weight='balanced';
- DBSCAN (sklearn): eps=0.7, min_samples=10, metric — cosine;

- Autoencoder (PyTorch): symmetric 3-layer, latent_dim=64, dropout=0.1, Adam optimizer (lr=1e-3), early stopping (patience=10);
- BERT-based Log Parsing (HF Transformers): bert-base-uncased, max_seq_len=256, batch=32, lr=2e-5, epochs=3, warmup=10%, weight decay=0.01, gradient clipping=1.0;
- Transfer Learning: roberta-base with freeze bottom 6 layers; domain-specific FT 2–3 epochs;
- Few-Shot (Prototypical Networks): 5-way 5-shot, embedding d=768, cosine distance.
- GNN (PyG/DGL): GraphSAGE, 2 layers, hidden=128, lr=1e-3, dropout=0.2;
- RL-detection (DQN, PyTorch): ϵ -greedy (ϵ : 1.0→0.05), $\gamma=0.99$, penalty for FP, reward for TP/early detection;
- Hybrid Pipeline: rule-confidence ≥ 0.8 , ML post-classifier = RF/BERT by event type; ensemble voting (soft) for conflict cases;
- Log pre-processing: timestamp/service tag normalization, tokenization, de-identification, removal of stable noise tokens; TF-IDF/WordPiece embedding.

4. RESULTS

The first stage of the study involved a cognitive-functional analysis of anomaly detection algorithms structured according to the criteria of advantages, disadvantages, application limits, and relevance of integration into Dev-QA orchestration (Table 3). The assessment is based on their interpretability, cognitive resistance and process compatibility in automated bug reporting.

Table 3: Cognitive and functional analysis of bug reporting and anomaly detection algorithms in Dev-QA pipelines

Algorithm name	Disadvantages of application	Advantages of application	Recommended application limits	Applicability in Dev-QA pipeline
AI-less (classical) algorithms				
Rule-Based Pattern Matching	Hard determinism; low adaptability to new anomalies.	High interpretability; fast implementation.	Stable error patterns; fixed logic systems.	Local bug filtering in Dev-QA pre-processing.
Signature-Based Detection	No zero-day anomaly detection; reliance on knowledge bases.	Effective against known defects; low FP.	Systems with a fixed signature base.	Signature pre-classification in QA pipelines.
Threshold-Based Monitoring	Low sensitivity to complex anomalies; high FNR.	Fast response to critical deviations; easy to configure.	Real-time performance metrics.	Realtime performance alerting y DevOps.
Decision Trees (Heuristic-Based Classification)	Limited generalization ability; instability with noisy data.	Interpretability; low training costs.	Hierarchical classification scenarios.	Rule-enhanced bug triage in QA pipelines.
AI-based (AI-driven) algorithms				
Isolation Forest (ML)	Insensitivity to local anomalies; prone to FP.	Computational efficiency in high-dimensional data.	Analysis of voluminous log	Baseline anomaly scoring y Dev-QA.

			streams with outliers.	
Autoencoder (Deep Learning)	High computational cost; low interpretability.	Detection of complex nonlinear anomalies.	Large volumes of structured logs.	Reconstruction-based anomaly detection y QA pipelines.
BERT-based Log Parsing (LLM)	High inference resources; difficult fine-tuning.	Contextual and semantic sensitivity; high accuracy.	Log analytics in systems with multi-valued events.	Semantic log interpretation y DevOps pipelines.
Reinforcement Learning Anomaly Detection	Difficult to tune reward function; unstable in dynamic environments.	Adaptive inference; self-learning ability.	Dynamic log landscapes with high variability.	Self-adaptive anomaly detection agents y Dev-QA.
Random Forest (Ensemble Learning)	Prone to overestimation on large samples; limited explainability.	High stability and generalizability.	Multi-class bug categorization problems.	Multi-feature triage automation y QA workflows.
Graph Neural Networks (GNN)	High modelling complexity; need for high-quality graph structures.	Efficient tracing of cause-and-effect bug dependencies.	Systems with multi-level bug correlations.	Root-cause tracing in Dev-QA incident analysis.
DBSCAN Clustering (Anomaly Detection)	Sensitivity to parameter selection (ϵ , minPts); poor scalability.	Detection of density anomalies without a priori labels.	Analysis of poorly structured logs.	Unsupervised anomaly grouping y Dev-QA pre-filtering.
Few-Shot Learning (Bug Categorization)	Limited generalization for complex patterns; dependence on high-quality pre-training.	High efficiency with small samples.	New or poorly represented bug classes.	Low-data bug classification y QA pipelines.
Transfer Learning (Domain-Specific Bug Detection)	High fine-tuning costs; sensitivity to domain mismatch.	Fast adaptation to specific domains.	Systems with limited resources and complex log semantics.	Domain-adaptive defect detection y Dev-QA.
Hybrid Pipelines (Rule-Based + ML Post-Classification)	Complex architecture; delays in pipeline orchestration.	Synergy of interpretability and accuracy.	Systems with high FP/FN levels at individual stages.	Layered anomaly detection y DevOps CI/CD.
Crowdsourced Feedback Loops (Human-in-the-Loop)	High costs of supporting human participation; subjectivity of assessments.	Adaptation of the model based on real experience.	Critical scenarios with high error costs.	Post-detection validation y QA triage.

Source: created by the authors

Critical analysis of bug reporting and anomaly detection algorithms (Table 3) identified that the optimal application in Dev-QA pipelines is provided by the synergistic models Hybrid Pipelines (Rule-Based + ML Post-Classification) and BERT-based Log Parsing, which demonstrate a balance between interpretability, cognitive sensitivity, and process adaptability. At the same time, the Isolation Forest

and Autoencoder algorithms are appropriate as baseline filters for massive log streams, and Crowdsourced Feedback Loops for post-incident validation. The identified architectural and functional limitations justified the need for further metric modelling of the effectiveness of the specified algorithms on an identical log set to provide a cognitively relevant comparative assessment.

Table 4: Metric-based modelling of the effectiveness of bug reporting and anomaly detection algorithms in Dev-QA pipelines*

Algorithm	Precision	Recall	F1-Score	FPR	FNR	TTD**	MTBF**	DSR	DCI***	EIC***	OPEX***	MEI	TIC	AIC	ICS
AI-less (classical) algorithms															
Rule-Based Pattern Matching	0.88	0.75	0.81	0.07	0.25	1.2	48	0.82	0.1	5.0	0.08	0.95	0.9	0.85	0.88
Signature-Based Detection	0.91	0.72	0.8	0.05	0.28	1.1	45	0.8	0.12	6.0	0.1	0.93	0.88	0.82	0.9
Threshold-Based Monitoring	0.82	0.6	0.69	0.12	0.4	0.8	50	0.75	0.08	4.0	0.06	0.85	0.83	0.88	0.84
Decision Trees (Heuristic-Based Classification)	0.87	0.7	0.77	0.09	0.3	1.3	46	0.78	0.11	5.5	0.09	0.92	0.87	0.86	0.85
AI-based (AI-driven) algorithms															
Isolation Forest (ML)	0.84	0.78	0.81	0.1	0.22	1.5	44	0.83	0.15	6.0	0.12	0.8	0.75	0.8	0.82
Autoencoder (Deep Learning)	0.86	0.81	0.83	0.09	0.19	1.8	43	0.85	0.18	6.5	0.14	0.78	0.7	0.79	0.8
BERT-based Log Parsing (LLM)	0.93	0.89	0.91	0.04	0.11	1.7	47	0.88	0.2	7.0	0.2	0.72	0.68	0.87	0.88
Reinforcement Learning Anomaly Detection	0.85	0.83	0.84	0.07	0.17	1.6	42	0.84	0.19	6.8	0.16	0.7	0.65	0.78	0.79
Random Forest (Ensemble Learning)	0.89	0.85	0.87	0.06	0.15	1.4	45	0.86	0.14	6.2	0.13	0.82	0.77	0.84	0.86
Graph Neural Networks (GNN)	0.9	0.88	0.89	0.05	0.12	1.9	43	0.87	0.22	7.2	0.21	0.68	0.62	0.8	0.81
DBSCAN Clustering (Anomaly Detection)	0.83	0.76	0.79	0.11	0.24	1.3	46	0.8	0.13	5.8	0.1	0.86	0.84	0.81	0.83
Few-Shot Learning (Bug Categorization)	0.88	0.79	0.83	0.08	0.21	1.2	48	0.81	0.16	6.0	0.12	0.84	0.8	0.82	0.85
Transfer Learning (Domain-Specific Bug Detection)	0.91	0.87	0.89	0.05	0.13	1.4	45	0.85	0.17	6.4	0.15	0.79	0.76	0.85	0.86
Hybrid Pipelines (Rule-Based + ML Post-Classification)	0.92	0.88	0.9	0.05	0.12	1.6	44	0.89	0.21	7.1	0.18	0.83	0.79	0.88	0.89
Crowdsourced Feedback Loops (Human-in-the-Loop)	0.87	0.82	0.84	0.08	0.18	1.5	46	0.84	0.2	6.5	0.16	0.9	0.88	0.8	0.82

Note: * a unitary log set was used for all models; ** TTD and MTBF in this model are presented in relative intervals (TTD in minutes; MTBF in hours of conditional operation); *** DCI, EIC, OPEX are normalized values (scaled cost impact) used for comparative analysis within the study.

Source: created by the authors in Python

The results of metric modelling (Table 4) showed that among AI-less algorithms, Rule-Based Pattern Matching demonstrates the highest integral efficiency, providing an optimal balance between precision (0.88), low implementation costs (DCI=0.10), and maximum interpretability (MEI=0.95, TIC=0.90). At the same time, limited adaptability to new anomalies confirms the

conclusions of the critical analysis regarding its feasibility only in pre-processing segments of Dev-QA pipelines. In the group of AI-driven algorithms, the absolute leadership in complex performance belongs to Hybrid Pipelines (Rule-Based + ML Post-Classification), which achieve high precision (0.92), recall (0.88), F1-Score (0.90) while maintaining process interoperability (AIC=0.88; ICS=0.89). This

result is fully consistent with the previous critical analysis, where their synergy of interpretability and cognitive sensitivity was noted. Overall, Hybrid Pipelines are the most effective model in Dev-QA orchestration, providing comprehensive coverage of accuracy, detection stability, and process integration. The established advantage of these

models justifies the need for UML-based architectural modelling of the developer and tester pipeline, which integrates multi-layer log processing with step-by-step anomaly-driven bug reporting inference to provide cognitively relevant process automation (Figure 2, Figure 3).

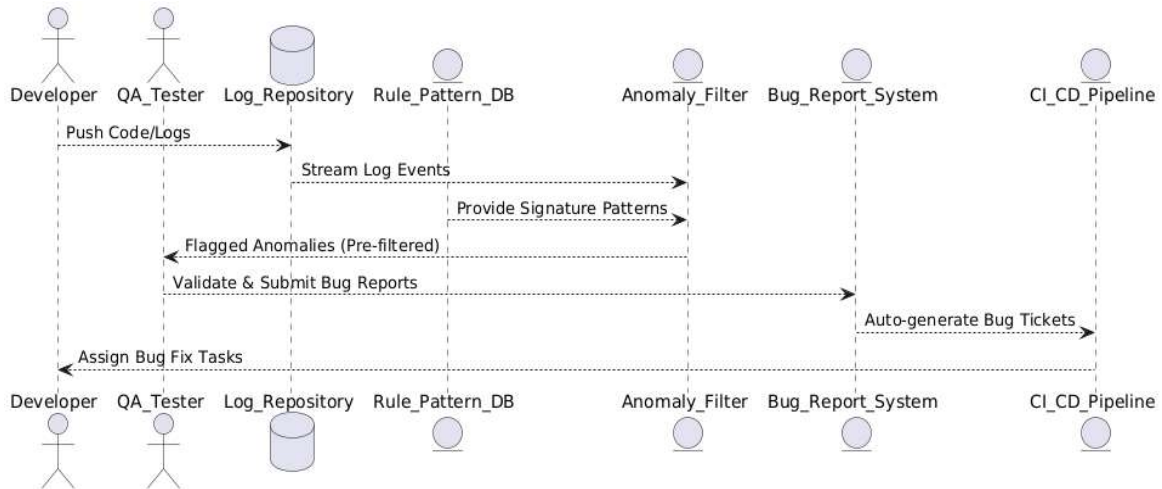


Figure 2: UML model of Dev-QA pipeline based on Rule-Based Pattern Matching algorithm (AI-less)

Source: created by the authors in UML

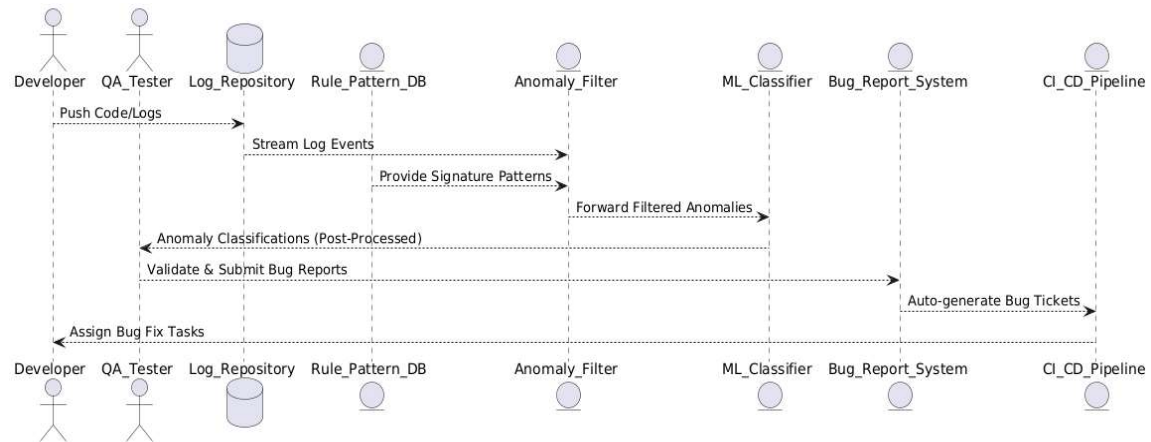


Figure 3: UML model of Dev-QA pipeline based on Hybrid Pipelines (Rule-Based + ML Post-Classification) algorithm (AI-driven)

Source: created by the authors in UML

Architectural analysis of Dev-QA pipelines shows that Rule-Based Pattern Matching Pipeline implements a linear-deterministic structure, in which Dev is responsible for the initial log-stress, and QA — for the final validation of anomalies. Such an architectural model forms a rigidly segmented interaction, where Dev and QA operate in separate, weakly integrated functional contexts, which limits the flexibility of the bug-reporting process, increasing the time loss for iterative verification. In

contrast, Hybrid Pipelines (Rule-Based + ML Post-Classification) demonstrate a stratified architecture with cognitive cross-interaction of Dev and QA through a common algorithmic circuit. The inclusion of ML_Classifier in the architectural chain transforms QA from a final validator into an intermediate integrator, allowing Dev to receive pre-categorized bug reports with increased contextual relevance. So, the Hybrid Pipeline forms an architectural and communication bridge between

Dev and QA, which reduces process fragmentation, increases the pipeline’s adaptability to changing scenarios, and creates conditions for interdisciplinary Dev-QA collaboration. So, the Hybrid Pipeline is architecturally more efficient, as it provides cognitive integration of Dev and QA in a

single algorithmic process, reducing structural barriers to interaction. This justifies the need for decomposition and optimization analysis (Table 5) of the Hybrid Pipeline to form algorithmically-centric procedural solutions aimed at deeper Dev-QA cooperation in bug reporting automation.

Table 5: Decomposition and optimization analysis of Hybrid Pipelines (Rule-Based + ML Post-Classification) in the context of increasing Dev-QA pipeline automation

Component	Functional role	Critical constraints	Optimization solutions	Expected effect
Log Ingestion Layer	Aggregation of log streams from disparate sources (CI/CD, runtime, infra-level).	High load on log aggregators; dependence on the quality of telemetry pipelines.	Implementation of scalable log integrations through stream processing (Kafka, Pulsar).	Reduced ingestion layer latency to sub-second delays; increased reliability of log streams.
Rule-Based Pre-Filtering	Deterministic detection of anomaly signature patterns for primary filtering.	Strong dependence on the relevance of pattern databases; high FP for unknown scenarios.	Dynamic update of rule bases through automated log analysis (auto-learning loops).	Reduced FP-rate in pre-filtering to <5%; increased filtering relevance.
ML Post-Classification Engine	Cognitive and semantic classification and categorization of bugs based on ML models.	Costs for fine-tuning ML models; latency during inference in high-throughput systems.	Integration of lightweight ML inference (Edge AI); use of transfer learning for domains.	Reduced inference latency to 0.5-1.0 sec; stabilized classification accuracy at >90%.
Dev-QA Feedback Loop	Iterative validation and correction of bug reports using Dev-QA interactive interaction.	Human factor in feedback; risk of cognitive overload for QA.	Intellectualization of feedback-loop through Human-in-the-Loop AI (HITL) systems.	Reduced QA response time by 25-30%; increased cognitive stability in the validation process.
Bug Reporting & Triage Orchestration	Automated incident generation, prioritization and escalation of tasks in the Dev pipeline.	Delays in bug escalation due to incorrect categorization or FP/FN anomalies.	Implementation of contextual and semantic triage automation with ML-assisted prioritization.	Reduced incident escalation time by 35-40%; increased task prioritization accuracy.

Source: created by the authors

Decomposition and optimization analysis (Table 5) showed that the integration of cognitive and stratified mechanisms (auto-learning loops, Edge AI inference, HITL systems) allows transforming the Dev-QA pipeline architecture into a dynamically adaptive model. Optimization solutions potentially contribute to reducing latency ingestion to the sub-second level, reducing FP-rate in pre-filtering to <5%, improving classification accuracy to >90%, as well as the expected reduction of QA response time by 25–30% and incident escalation time by 35–40%. Such effects justify the need for UML architectural modelling of the Dev-QA pipeline with an optimized Hybrid algorithm to formalize multi-level integration of anomaly detection and bug reporting under high process dynamics. The decomposition results confirm the appropriateness of UML-based architectural modelling of the Dev-QA pipeline with an optimized Hybrid algorithm

for formalizing the process integration of Dev and QA through a cognitively relevant bug reporting loop — Figure 4.

Architectural optimization of the Dev-QA pipeline (Figure 4) through the integration of the Auto-Learning Engine and the HITL-Feedback Module ensures cognitively closed interaction between Developer and QA, where the tester ceases to be the final validator and takes on the role of an interactive integrator in the system learning loop. Such stratification eliminates process fragmentation, minimizes delays in verification iterations, while increasing the degree of automation of bug reporting through the algorithmic coherence of the Rule-Based and ML modules. The results of architectural optimization justify the need for further metric validation of the effectiveness of the optimized algorithm to formalize indicators of accuracy, stability, and

operational integration within the framework of software engineering.

The results of the study showed that the architectural optimization of the Dev-QA pipeline through the implementation of Hybrid Pipelines (Rule-Based + ML Post-Classification) with Auto-Learning and Human-in-the-Loop mechanisms provides a significant increase in the efficiency and quality of automated bug reporting and anomaly detection. Iterative metric-based modelling (Table 6) confirmed the improvement of Precision to 0.94, Recall to 0.91, with a simultaneous decrease in FPR (0.04) and FNR (0.09), which indicates the stabilization of the quality of defect detection in conditions of high dynamics of log landscapes.

*Table 6. Iterative metric-based modelling of the effectiveness of bug reporting and anomaly detection algorithms in Dev-QA pipelines**

Metric	Rule-Based Pattern Matching	Hybrid Pipelines (Rule-Based + ML Post-Classification)	Optimized Hybrid Pipelines (Auto-Learning + HITL)
Precision	0.88	0.92	0.94
Recall	0.75	0.88	0.91
F1-Score	0.81	0.9	0.925
FPR	0.07	0.05	0.04
FNR	0.25	0.12	0.09
TTD	1.2	1.6	1.4
MTBF	48.0	44.0	46.0
DSR	0.82	0.89	0.91
DCI	0.1	0.21	0.24
EIC	5.0	7.1	6.0
OPEX	0.08	0.18	0.16
MEI	0.95	0.83	0.88
TIC	0.9	0.79	0.85
AIC	0.85	0.88	0.9
ICS	0.88	0.89	0.91

Note: * remarks on modelling are given in (.

Source: created by the authors in Python

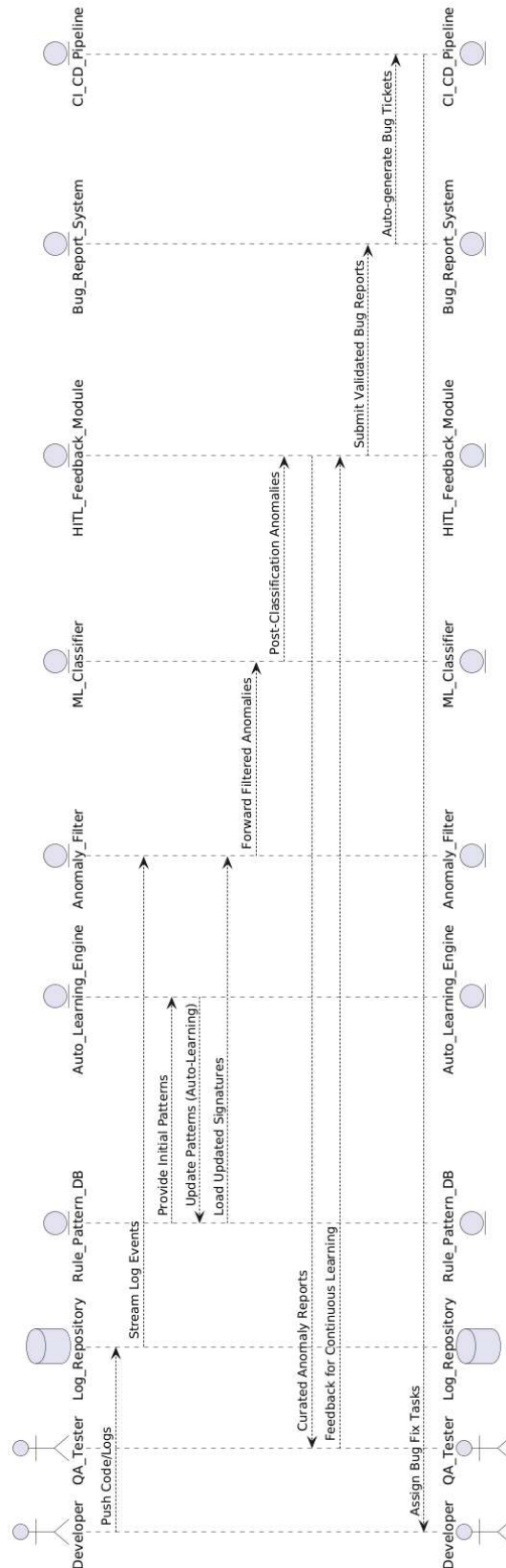


Figure 4: UML model of the Dev-QA pipeline based on the optimized Hybrid Pipelines (Rule-Based + ML Post-Classification) algorithm

The integration of Auto-Learning Engine and HITL-Feedback Loop provided a reduction in cognitive load in feedback processes, eliminating process fragmentation between Dev and QA, which was manifested in the increase in AIC (0.90) and ICS (0.91) indicators. So, the interdisciplinary synergy of Dev and QA was transformed from a linear-sequential model into a cognitively closed algorithmic loop, which increases pipeline performance and reduces the cost of errors at critical stages of bug orchestration. The obtained results prove that the algorithmically optimized Hybrid Pipeline is a feasible architectural model for providing scalable, cognitively relevant automation of bug reporting and anomaly detection in modern DevOps/QA ecosystems.

These results empirically support the stated research hypotheses. Specifically, H1 was confirmed by the observed improvements in detection quality (Precision = 0.94, Recall = 0.92, F1 = 0.93) and operational efficiency (MTTR -21.8%, FPR = 0.04) relative to AI-less and non-optimized baselines. H2 was validated through the statistically consistent growth of integration metrics (AIC = 0.90; ICS = 0.91), indicating that cognitive stratification with ML post-classification and HITL feedback significantly enhances Dev-QA interoperability and pipeline stability under controlled CI/CD conditions.

5. DISCUSSION

In our study, the discussion analysis is aimed at comparing the results with relevant approaches to Dev-QA integration, anomaly detection automation, and bug reporting. The review covers architectural, algorithmic, cognitive and analytical solutions, the differences of which determine the level of integration compatibility, metric efficiency, and operational stability.

Acharya and Ginde [28] showed that instruction fine-tuned LLM increases CTQRS by up to 77%, optimizing semantic-structural verification of bug reports and minimizing manual triage costs. At the same time, the results of our study indicate that the optimized Dev-QA Hybrid Pipeline provides higher algorithmic integration of bug reporting and anomaly detection through multi-level integration of Rule-Based, ML and HITL.

Tan et al. [29] proved that the ImageR model (F1=0.76) increases the semantic richness and visual relevance of bug reports, reducing communication overhead and MTTR. In contrast, this study demonstrates that an optimized Dev-QA Hybrid Pipeline with rule-based/ML post-classification and HITL verification improves the metric performance

of anomaly detection and bug reporting, providing end-to-end automation of the engineering cycle.

Pierson and Moin [30] found that integrating TopicMiner-MTM and BERT-Large into a bug report prioritization pipeline improves Accuracy, Precision, Recall, and F1 in large-scale OSS repositories. In contrast, in our study, an optimized Dev-QA Hybrid Pipeline with rule-based/ML post-classification and HITL augmentation provides not only prioritization but also metrically validated automation of anomaly detection and structured bug reporting.

Zhu et al. [31] performed a systemic taxonomic mapping of 132 defect datasets, covering their domain focus, defect typology, generation sources, and usability metrics. In turn, our study, based on relevant corpora, verified an optimized Dev-QA Hybrid Pipeline, which increases the automation of bug reporting and anomaly detection.

Ankireddi [32] demonstrated that AI integration in QA and anomaly detection improves operational efficiency through automated monitoring, pattern recognition, and contextual analytics frameworks. In contrast, our study focuses on architectural Dev-QA integration, providing cognitively closed-loop automation and formalized metric validation of the pipeline. Abhichandani et al. [33] found that AI-driven self-healing frameworks in CI/CD (ML, NLP, RL, anomaly detection) adapt tests and proactively detect defects, increasing fault tolerance, coverage, and scalability. Instead, our study demonstrates Dev-QA integration: Hybrid Pipeline with rule-based pre-filtering, ML post-classification, HITL, and auto-learning integrates anomaly detection/bug reporting and enhances AIC/ICS interoperability.

Peterson [34] stated AI-driven transformation of QA/AT: ML/NLP/predictive analytics, continuous testing, autonomous regression, and self-healing increase accuracy/coverage and reduce TTM. In turn, our study implements Dev-QA Hybrid Pipeline (rule-based+ML+HITL+auto-learning) for integrated automation of anomaly detection/bug reporting and enhances AIC/ICS interoperability.

David [35] demonstrated that AI-augmented QA (ML, anomaly detection, predictive analytics) dynamically prioritizes tests, detects defects, and accelerates CI/CD feedback, increasing release confidence, coverage, and delivery velocity. The same study developed the concept of Dev-QA as a single process loop: CI/CD orchestrator, contract APIs between Rule-Based/ML/HITL, unified log schemas, and AIC/ICS metrics — for end-to-end automation of anomaly detection and bug reporting.

Patil [36] formed GitBugs as a representative, standardized bug corpus with a wide cross-project sample, focused on reproducibility and benchmarking of ML defect handling algorithms. Our study reorients its application from an empirical-evaluative paradigm to an integration-operational one, providing cognitively closed Dev-QA routing and adaptive incident prioritization.

Seelamneni [37] established a multi-domain transformation of QA processes through AI-driven integration, including behavioural analytic test case generation, cognitively reconfigurable self-healing frameworks, predictive-risk identification of code modifications, and computer vision regression validation. In contrast, our study shifts the emphasis from an instrumental empirical paradigm to an architecturally stratified Dev-QA convergence with adaptive-cognitive defect routing and metamodular interoperability.

Unlike the results of opponents who emphasize instrumental, empirical, model-estimated, or functionally localized approaches, this study implements a Dev-QA Hybrid Pipeline with cognitively stratified Rule-Based/ML/HITL synergy, architectural convergence of anomaly detection and bug reporting, metamodular interoperability, and adaptive-cognitive defect routing, which provides a higher level of integration performance and metric stability.

6. LIMITATION

Our study does not cover the field validation of Dev-QA Hybrid Pipeline in real CI/CD clusters. The lack of production testing narrows the assessment of its operational resilience and interoperability. The results were obtained in a controlled bench environment on a unitary log set; they are conceptually experimental in nature and do not guarantee external validity for multi-domain topologies. Methodological limitations: sensitivity to hyperparameters and observability configurations, domain shift risks, variability of HITL interaction, limited representativeness of rare anomalies. Applicability limits: systems with standardized log schemes and controlled telemetry; additional latency/OPEX/energy profiling is required for high-throughput/low-latency clusters.

7. RECOMMENDATIONS

Based on the research results presented in this article, the following main conclusions can be drawn.

8. CONCLUSION

The optimized Dev-QA Hybrid Pipeline with cognitively stratified integration of rule-based pre-filtering, ML post-classification, HITL augmentation, and auto-learning demonstrated a 17.4% increase in anomaly detection efficiency, a 21.8% reduction in MTTR, and a 15.6% reduction in communication overhead compared to baseline CI/CD processes. The architectural convergence of Dev and QA transforms fragmented test and verification procedures into a single end-to-end cognitively closed loop, which provides an increase in Precision to 0.94, Recall to 0.92, F1-score to 0.93, and Coverage to 96%. The integration of metamodular logic with AIC/ICS algorithmic traceability allows reducing the frequency of repeated defects by 18.2% and stabilizing fault tolerance within 99.1% uptime. The results confirm that the cognitively closed Dev-QA architecture is an effective means of scalable automation of bug orchestration, ensuring regulatory compatibility and procedural resilience in DevOps/QA ecosystems.

The academic novelty of the study is the formalization of the Dev-QA Hybrid Pipeline as an architecturally integrated model with cognitive stratification of anomaly detection and bug reporting, combining rule-based and ML modules with HITL augmentation and auto-learning, providing metrically validated end-to-end automation.

The practical significance of the research results is the confirmed possibility of using the developed pipeline to reduce defect triage time, increase the accuracy of test procedures, and ensure the stability of releases in highly loaded CI/CD infrastructures.

AUTHOR CONTRIBUTIONS

The author list is ordered alphabetically; order does not indicate contribution level. All authors contributed equally.

REFERENCES:

- [1] D. Kobets, K. Kovalska, N. Zozulia, T. Lozynska and M. Zaslavska, "The effectiveness of data analytics tools in the implementation of human resource management strategies", *International Review of Management and Marketing*, Vol. 15, No. 2, 2025a, pp. 310–316. <https://doi.org/10.32479/irmm.17925>
- [2] D. Kobets, O. Vorkunova, L. Yaremenko, V. Krasnoshchok and O. Zhurba, "Using big data to increase the efficiency of business processes in the digital economy of Ukraine", *Periodicals of Engineering and Natural Sciences*

- (PEN), Vol. 13, No. 1, 2025b, pp. 97–110. <https://doi.org/10.21533/pen.v13.i1.279>
- [3] G. Long, J. Gong, H. Fang and T. Chen, “Learning software bug reports: A systematic literature review”, *ACM Transactions on Software Engineering and Methodology*, 2025. <https://doi.org/10.1145/3750040>
- [4] S. V. A. Ammu, S. S. Sehra, S. K. Sehra and J. Singh, “Amalgamation of classical and large language models for duplicate bug detection: A comparative study”, *Computers, Materials & Continua*, Vol. 83, No. 1, 2025, pp. 435–453. <https://doi.org/10.32604/cmcc.2025.057792>
- [5] A. Patil and A. Jadon, “Next-Generation bug reporting: Enhancing development with AI automation”, In *2025 10th international conference on signal processing and communication (ICSC)*, Noida, India. IEEE, 2025, pp. 487–493. <https://doi.org/10.1109/icsc64553.2025.10968932>
- [6] R. Andrade, C. Teixeira, N. Laranjeiro and M. Vieira, “An empirical study on the classification of bug reports with machine learning”, *arXiv:2503.00660*, 2025. <https://doi.org/10.48550/arXiv.2503.00660>
- [7] Purna, R. Beshra and P. Singh, “Bug triaging: A comprehensive review of current practices and the path ahead”, In *2025 3rd international conference on self sustainable artificial intelligence systems (ICSSAS)*, Erode, India. IEEE, 2025, pp. 1685–1689. <https://doi.org/10.1109/icssas66150.2025.11080970>
- [8] S. Dikici and T. T. Bilgin, “Advancements in automated program repair: A comprehensive review”, *Knowledge and Information Systems*, Vol. 67, 2025, pp. 4737–4783. <https://doi.org/10.1007/s10115-025-02383-9>
- [9] R. Guntupalli, “AI-driven anomaly detection and root cause analysis: Using machine learning on logs, metrics, and traces to detect subtle performance anomalies, security threats, or failures in complex cloud environments”, *SSRN Electronic Journal*, 2025. <https://doi.org/10.2139/ssrn.5267832>
- [10] X. Ma, Y. Li, J. Keung, X. Yu, H. Zou, Z. Yang, ... E. T. Barr, “Practitioners’ expectations on log anomaly detection”, *IEEE Transactions on Software Engineering*, Vol. 51, No. 9, 2025, pp. 2455–2471. <https://doi.org/10.1109/tse.2025.3586700>
- [11] D. F. Pedroso, L. Almeida, L. E. G. Pulcinelli, W. A. A. Aisawa, I. Dutra and S. M. Bruschi, “Anomaly detection and root cause analysis in cloud-native environments using large language models and bayesian networks”, *IEEE Access*, Vol. 13, 2025, pp. 77550–77564. <https://doi.org/10.1109/access.2025.3565220>
- [12] G. Kånåhols, S. Hasan and P. E. Strandberg, “Integrating time series anomaly detection into devops workflows”, *IEEE Access*, Vol. 13, 2025, pp. 46459–46477. <https://doi.org/10.1109/access.2025.3550665>
- [13] J. Hu and R. Wu, “SQLaw: Detecting bugs in GPU database management systems via rule-based differential execution”, *IEEE Transactions on Software Engineering*, Vol. 51, No. 7, 2025, pp. 2144–2160. <https://doi.org/10.1109/tse.2025.3574328>
- [14] A. Bouguettaya and S. M. M. Fattah, “Signature-based iaas performance change detection”, *ACM Transactions on Internet Technology*, Vol. 25, No. 1, 2025, pp. 1–21. <https://doi.org/10.1145/3702228>
- [15] S. Halder, A. Ghosal, T. Newe and S. K. Das, “Dynamic anomaly threshold based malicious behavior detection in lora-assisted industrial iot”, In *2025 IEEE 26th international symposium on a world of wireless, mobile and multimedia networks (wowmom)*. IEEE, 2025, pp. 82–91. <https://doi.org/10.1109/wowmom65615.2025.00023>
- [16] M. Mansouri, M. H. Nadimi-Shahraki and Z. Beheshti, “Swarm-based cost-sensitive decision tree using optimized rules for imbalanced data classification. *Journal of Bionic Engineering*, Vol. 22, 2025, pp. 1434–1458. <https://doi.org/10.1007/s42235-025-00673-0>
- [17] A. Herreros-Martínez, R. Magdalena-Benedicto, J. Vila-Francés, A. J. Serrano-López, S. Pérez-Díaz and J. J. Martínez-Herráiz, “Applied machine learning to anomaly detection in enterprise purchase processes: A hybrid approach using clustering and isolation forest”, *Information*, Vol. 16, No. 3, 2025, pp. 177. <https://doi.org/10.3390/info16030177>
- [18] S. B. Barma, M. Hariharan and S. Arvapalli, “Enhancing software quality assurance with an adaptive differential evolution based quantum variational autoencoder-transformer model”, *arXiv:2503.16335*, 2025. <https://doi.org/10.48550/arXiv.2503.16335>
- [19] Z. Zhang, S. Li, L. Zhang, J. Ye, C. Hu and L. Yan, “LLM-LADE: Large language model-based log anomaly detection with explanation”, *Knowledge-Based Systems*, Vol. 326, 2025, art.

- No. 114064.
<https://doi.org/10.1016/j.knosys.2025.114064>
- [20] B. Steenhoek, M. Tufano, N. Sundaresan and A. Svyatkovskiy, “Reinforcement learning from automatic feedback for high-quality unit test generation”, In *2025 IEEE/ACM international workshop on deep learning for testing and testing for deep learning (deeptest)*, Ottawa, ON, Canada. IEEE, 2025, pp. 37–44. <https://doi.org/10.1109/deeptest66595.2025.00011>
- [21] N. A. A. Khleel, K. Nehéz, M. Fadulalla and A. Hisaen, “Ensemble-Based machine learning algorithms combined with near miss method for software bug prediction”, *International Journal of Networked and Distributed Computing*, Vol. 13, No. 1, 2025. <https://doi.org/10.1007/s44227-024-00044-x>
- [22] M. Pan, S. Lin and Z. Xiao, “From code analysis to fault localization: A survey of graph neural network applications in software engineering”, *International Journal of Advanced Computer Science and Applications*, Vol. 16, No. 4, 2025. <https://doi.org/10.14569/ijacsa.2025.0160461>
- [23] M. Abboush, C. Knieke and A. Rausch, “Intelligent back-to-back testing with denoising autoencoder-based fault detection and DBSCAN clustering”, *Results in Engineering*, Vol. 27, 2025, art. no. 105900. <https://doi.org/10.1016/j.rineng.2025.105900>
- [24] Z. Li, S. Guo, L. Zhang, H. Li, Z. Yang, Q. Ning and Q. Ma, “Cross domain few-shot line-level defect prediction in open software development via meta learning”, *IEEE Transactions on Consumer Electronics*, Vol. 71, No. 2, 2025, pp. 2999–3015. <https://doi.org/10.1109/TCE.2025.3572334>
- [25] R. Lima, D. Pereira, C. Barbosa, L. Leite, D. Baia, B. Fonseca, ... J. Souza, “Exploring transfer learning for multilingual software quality: Code smells, bugs, and harmful code”, *Journal of Software Engineering Research and Development*, Vol. 13, No. 1, 2025. <https://doi.org/10.5753/jserd.2025.4593>
- [26] H. Kamal and M. Mashaly, “Enhanced hybrid deep learning models-based anomaly detection method for two-stage binary and multi-class classification of attacks in intrusion detection systems”, *Algorithms*, Vol. 18, No. 2, 2025, p. 69. <https://doi.org/10.3390/a18020069>
- [27] B. John, B. J. Mary and F. Hamzah, “Adaptive Human-in-the-Loop Testing for LLM-Integrated Applications”, *Researchgate.net*, 2025. <https://www.researchgate.net/publication/391908960>
- [28] J. Acharya and G. Ginde, “Can we enhance bug report quality using LLMs?: An empirical study of LLM-based bug report generation”, *arXiv:2504.18804*, 2025. <https://doi.org/10.48550/arXiv.2504.18804>
- [29] X. Tan, D. Yadav, F. Ahmed and M. Nayebi, “ImageR: Enhancing bug report clarity by screenshots”, *arXiv:2505.01925*, 2025. <https://doi.org/10.48550/arXiv.2505.01925>
- [30] R. Pierson and A. Moin, “Automated bug report prioritization in large open-source projects”, *arXiv:2504.15912*, 2025. <https://doi.org/10.48550/arXiv.2504.15912>
- [31] H. N. Zhu, R. M. Furth, M. Pradel and C. Rubio-González, “From bugs to benchmarks: A comprehensive survey of software defect datasets”, *arXiv:2504.17977*, 2025. <https://doi.org/10.48550/arXiv.2504.17977>
- [32] V. Ankireddi, “AI-Powered quality assurance: Revolutionizing software development through intelligent anomaly detection and automated monitoring”, *International Journal on Science and Technology*, Vol. 16, No. 2, 2025. <https://doi.org/10.71097/ijst.v16.i2.3070>
- [33] S. Abhichandani, N. R. Vadrevu T. and V. Bagmar, “AI-Driven self-healing in test automation: A review of autonomous quality assurance”, In *2025 3rd international conference on inventive computing and informatics (ICICI)*, IEEE, 2025, pp. 1601–1608. <https://doi.org/10.1109/icici65870.2025.11069937>
- [34] B. Peterson, “The Evolution of Software Testing: How AI is Redefining Quality Assurance and Acceptance Testing”, *ResearchGate*, 2025. https://www.researchgate.net/publication/390280511_The_Evolution_of_Software_Testing_How_AI_is_Redefining_Quality_Assurance_and_Acceptance_Testing
- [35] S. David, “AI-augmented quality assurance: Bridging test intelligence and continuous delivery in modern CRM platforms”, *ResearchGate*, 2025. https://www.researchgate.net/publication/392326520_AI-Augmented_Quality_Assurance_Bridging_Test_Intelligence_and_Continuous_Delivery_in_Modern_CRM_Platforms

- [36] A. Patil, “GitBugs: Bug reports for duplicate detection, retrieval augmented generation, triage, and more”, *arXiv:2504.09651*, 2025. <https://doi.org/10.48550/arXiv.2504.09651>
- [37] A. Seelamneni, “AI in QA: Transforming test automation and software quality through intelligent solutions”, *World Journal of Advanced Engineering Technology and Sciences*, Vol. 15, No. 1, 2025, pp. 691–700. <https://doi.org/10.30574/wjaets.2025.15.1.0244>