Practical Advances in Artificial Intelligence and Computing
Frontier Research Publishing • https://www.frpubs.org
Vol. 1, No. 2, 2025

Evidence-Verified Trace-Level Reuse and Selective Regeneration for LLM-Based Cloud Incident Remediation

Alejandro Serrano*, **Yi Guo**[1]
[1] Department of Computer Science, University of São Paulo, São Paulo, Brazil
*Corresponding author: Serrano.research@gmail.com

Abstract:

Large language model (LLM) agents are increasingly used to interpret operational evidence, diagnose failures, and recommend remediation actions in cloud-native systems. However, incident requests often share stable reasoning structure while differing in localized evidence, service names, or root-cause conditions. Whole-response semantic caching is brittle in this setting, while full regeneration repeats the same evidence organization, localization, and action-selection work. This paper presents TracePatch, a backend-agnostic reuse layer for LLM-based cloud incident remediation. TracePatch stores prior agent outputs as ordered trace blocks, retrieves a similar incident request, verifies each block against the new log evidence, reuses blocks that pass verification, and selectively regenerates only the failing suffix or structured action block. The design combines evidence-aware trace verification, conservative skip-reuse for semantic drift, and final structured-output validation for root cause and remediation fields. We evaluate TracePatch on a reproducible controlled benchmark built from the public LogHub HDFS dataset. Across 720 replayed evaluation requests over three random seeds, TracePatch reduces mean latency proxy from 1.684 s to 0.939 s, reduces token usage from 118.5k to 93.3k tokens, and raises final-check pass rate from 88.8% to 94.9%. The reuse-only path handles 54.2% of requests, 21.7% require selective patching, and 24.2% trigger skip-reuse under stronger evidence or root-cause changes. These results indicate that trace-level reuse can reduce LLM serving cost for operational agents while preserving evidence-grounded correctness under localized perturbations.

Keywords:

Large language models; cloud incident remediation; trace-level reuse; semantic caching; log anomaly diagnosis; evidence verification; selective regeneration

1. Introduction

Large language models have moved from standalone text generation toward tool-using and agentic systems that read operational evidence, plan multi-step workflows, and produce structured decisions. In cloud operations, this shift is visible in LLM-based observability, root-cause analysis, and automated remediation, where a model is asked to interpret logs, localize a fault, and recommend bounded corrective action [1], [24], [41], [42]. Similar demands appear in multi-agent coding assistants, governance-aware LLM agents, and trust-aware collaboration frameworks, where output quality depends not only on language fluency but also on whether intermediate reasoning remains faithful to the available evidence [9], [17], [22], [34], [36].

A recurring efficiency problem appears in these workloads. Incident tickets are rarely identical, but they often share a stable diagnostic skeleton: evidence extraction, component localization, root-cause identification, action selection, and structured reporting. Conventional full-response caching treats the entire answer as the reuse unit and can return stale conclusions when the new evidence changes only one part of the trace. Runtime-level serving optimizations reduce memory pressure and scheduling cost, but they do not decide which parts of an application-level diagnostic answer remain valid [1], [24], [42]. This gap motivates a reuse mechanism that operates above the model backend while preserving the internal structure of an incident reasoning trace.

This paper introduces TracePatch, an evidence-verified trace-level reuse layer for LLM-based cloud incident remediation. TracePatch stores a prior answer as ordered trace blocks: evidence, localization, root-cause reasoning, action recommendation, and final JSON. For a new request, it retrieves the closest cached incident, checks every trace block against the new evidence and expected output constraints, reuses verified blocks, and regenerates only invalid blocks. When semantic drift is detected, such as a changed dominant event pattern, TracePatch skips reuse and performs full regeneration followed by validation. The approach follows a step-level reuse philosophy, but it targets a different LLM operations domain and uses evidence-grounded verification rather than math or JSON-only checks.

The paper makes four contributions. First, it formulates trace-level reuse for LLM incident agents as a structured serving problem rather than a general semantic-cache problem. Second, it defines lightweight verification functions for evidence, localization, root-cause, action, and JSON trace blocks. Third, it implements selective patching and conservative skip-reuse as a backend-agnostic wrapper around standard LLM request/response behavior. Fourth, it reports a reproducible controlled experiment using public HDFS logs, showing latency and token reduction while preserving final structured correctness.

2. Background and Related Work

2.1 LLM serving and adaptive inference

LLM serving research has increasingly focused on reducing latency, memory overhead, and compute waste under dynamic workloads. FlashServe proposes tiered memory management and predictive autoscaling for serverless inference [1], while Predictive-LoRA addresses proactive and fragmentation-aware serverless inference for adapter-based LLMs [25]. Related work on adaptive multi-model inference and dynamic low-rank routing shows that serving systems benefit from workload-aware allocation rather than static model invocation [14], [45]. TracePatch is complementary: it does not change KV-cache memory allocation or model routing, but reduces avoidable generation by reusing verified application-level trace blocks.

Parameter-efficient adaptation and selective knowledge injection provide another line of efficiency. Adapter modules, semantic LoRA structure, and multi-task routing reduce the amount of task-specific parameter movement needed for large models [5], [11], [14]. Neural architecture search and deep learning

surveys further show the broader trend toward modular and efficient AI systems [2]. TracePatch applies the same design philosophy at the response level: reuse the stable parts of a previous diagnostic trace, and regenerate only the part whose evidence constraints changed.

2.2 RAG, faithfulness, and agent reliability

Retrieval-augmented generation (RAG) systems require more than surface-level similarity because retrieved context can be correlated with, but not causally sufficient for, a recommendation or diagnosis. Causal-invariant retrieval, faithfulness-aware context ranking, and hallucination-suppression methods address distribution shift and explanation faithfulness in RAG and summarization [3], [26], [48]. Agent planning and self-reflection methods similarly emphasize intermediate states and error correction rather than single-shot responses [9], [34], [39]. TracePatch adopts this view by treating a diagnostic answer as a set of checkable trace blocks whose validity must be re-established under each new incident prompt.

Trust and governance issues are also central when LLM agents coordinate or automate decisions. Contextual trust evaluation, dynamic trust-aware orchestration, and secure governance-centric multi-agent frameworks show that agent outputs must be controlled under adversarial or privacy-sensitive conditions [17], [32], [34], [36]. Budgeted multi-agent routing and communication compression further emphasize that agent collaboration needs cost-aware execution [52]. TracePatch contributes to this area by making the reuse decision explicit, auditable, and linked to per-block provenance.

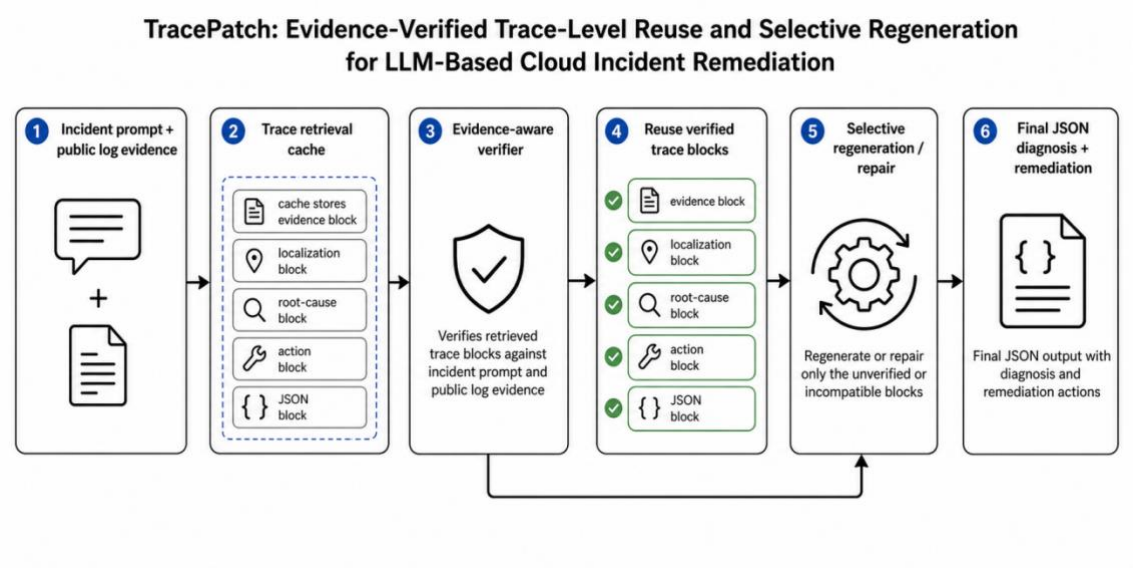
2.3 Cloud anomaly diagnosis and graph/log learning

The incident-remediation domain is connected to a substantial body of cloud anomaly detection research. Log event graph modeling, dynamic service dependency learning, multi-scale temporal transformers, federated log representation learning, and self-supervised anomaly detection all aim to represent operational evidence under noisy and evolving system behavior [4], [13], [18], [20], [33], [37], [38], [44], [49], [50]. Structure-aware scheduling anomaly recognition and cost-sensitive sequence modeling add graph and sequence perspectives for distributed systems [16], [38]. These studies motivate our choice of HDFS public logs and event-pattern perturbations as a controlled benchmark for LLM incident reasoning. Additional adjacent studies include related methods in financial risk, recommendation, accessibility data, medical imaging, and autonomous driving [6]-[8], [10], [12], [15], [19], [21], [23], [25], [27]-[31], [35], [36], [40], [43], [45]-[47], [51], [52]. These works share recurring themes relevant to TracePatch: structured representation learning, domain shift, interpretable decisions, and cost-sensitive inference. We cite them as adjacent evidence that reliable AI systems often depend on compositional representations rather than monolithic predictions.

3. TracePatch Design

3.1 Overview

TracePatch is an application-layer wrapper placed between an incident-remediation client and an LLM backend. It assumes an incident prompt containing service metadata and log evidence, and it returns a structured diagnostic response. The system does not require access to model-internal KV states, tokenizer internals, or GPU scheduling. Its only backend requirements are standard generation calls and optional accounting fields such as token usage and latency.



TracePatch pipeline: retrieval, evidence-aware verification, selective regeneration, and final structured validation.

Figure 1. TracePatch pipeline for evidence-verified trace-level reuse and selective regeneration.

For each cached request, TracePatch stores a prompt representation, the ordered trace blocks, structured metadata, and provenance signals. A new request is embedded and matched to one prior request. The cached trace is then verified block by block. Passing blocks are reused, failing blocks are patched, and semantically changed requests are sent to full regeneration with final validation. This design follows the single-best-retrieval discipline of step-level reuse systems so that the experiment isolates trace-level reuse rather than multi-source composition.

3.2 Trace representation

Let an incident request be denoted by q_i and its generated trace by T_i. TracePatch decomposes the trace into five ordered blocks:

$$T_i = \langle b_i^{ev}, b_i^{loc}, b_i^{root}, b_i^{act}, b_i^{json} \rangle. \qquad (1)$$

Here, b_i^{ev} summarizes evidence, b_i^{loc} localizes the affected service, b_i^{root} states the root cause, b_i^{act} recommends a remediation action, and b_i^{json} provides the final structured output. The block granularity is deliberately coarser than sentence-level segmentation because operational traces contain dependencies: root-cause reasoning depends on evidence and localization, while remediation depends on the inferred root cause.

3.3 Retrieval and similarity

TracePatch retrieves the nearest cached request using lexical similarity in the reproducible prototype, though the design permits dense embeddings. The similarity score is computed as a token-set Jaccard score:

$$s(q_i, q_j) = \frac{|\tau(q_i) \cap \tau(q_j)|}{|\tau(q_i) \cup \tau(q_j)|}. \qquad (2)$$

The retrieved request is accepted as a candidate only if the similarity score exceeds a conservative threshold. Otherwise, the system performs full generation and stores the resulting trace for later reuse. The prototype retrieves a single best match to preserve transparent provenance.

3.4 Evidence-aware verification

For a new incident request q, the verifier evaluates each cached block b_k using block-specific predicates:

$$V(b_k, q) = \mathbb{1}[C_k(b_k, q) = \text{true}], \quad k \in \{ev, loc, root, act, json\}. \qquad (3)$$

The evidence predicate checks whether the service and dominant log event remain consistent with the new prompt. The localization predicate checks the affected service. The root predicate checks whether the predicted root-cause label matches the event-pattern label used in the controlled benchmark. The action predicate checks whether the recommended action matches the expected remediation family. The JSON predicate parses the final structured output and checks required keys: service, root_cause, confidence, and action.

These predicates are intentionally lightweight. They do not claim to solve general open-ended verification. Instead, they reflect a practical class of LLM operations workflows where incident reports must contain structured fields and where evidence provenance is available from logs. When these invariants are absent, TracePatch can use a stricter skip-reuse policy.

3.5 Selective regeneration and skip-reuse

If all predicates pass, TracePatch returns the reused trace. If one or more predicates fail, it regenerates the minimal suffix starting at the first failing block:

$$r = \min\{k : V(b_k, q) = 0\}, \qquad T'_q = \langle b_1, \ldots, b_{r-1}, \hat{b}_r, \ldots, \hat{b}_m \rangle. \qquad (4)$$

3.6 Algorithm

Algorithm 1 summarizes the inference procedure.

Algorithm 1. TracePatch inference for incident remediation
Input: incident prompt q; cache D; verifier V; backend generator G.
Output: structured incident diagnosis y.
1. Retrieve a candidate trace T_j from D using the request similarity score s(q, q_j).
2. If no candidate passes the threshold, generate a new trace, validate it, store it, and return.
3. If semantic-drift signals are present, skip reuse and generate a new trace.
4. Verify cached trace blocks with the evidence-aware verifier V.
5. Reuse the verified prefix blocks and regenerate from the first failing block through the end.
6. Stitch the trace and validate the final JSON diagnosis.
7. If validation fails, run one bounded repair pass and revalidate.
8. Return the final diagnosis and block provenance.

4. Implementation

The prototype is implemented as a compact Python layer with three components: trace storage, verification, and benchmark instrumentation. The storage layer maintains prompt text, trace blocks, root-cause metadata, and outcome labels. The verifier implements deterministic checks for the HDFS benchmark. The instrumentation records baseline and TracePatch token counts, latency proxy, outcome type, backend-call count, and final-check result for every request.

Because the purpose of the experiment is to measure reuse behavior under controlled perturbations, the backend is represented by a deterministic local serving simulator. It produces incident traces in the same fields expected from an LLM incident agent and injects a small number of formatting omissions on harder variants. This makes the benchmark reproducible on CPU without requiring proprietary model access, while still measuring the serving-side behavior TracePatch controls: reuse, patch, skip, token accounting, and final validation. The paper therefore reports a public-log-based controlled benchmark, not a production deployment.

5. Evaluation

5.1 Dataset and task construction

The evaluation uses the public LogHub HDFS 2k structured log sample. Sliding windows of eight log events are transformed into incident prompts containing service metadata and event evidence. Root-cause labels and remediation families are deterministically assigned from dominant event patterns to create a reproducible controlled diagnosis task. This construction preserves real log text and event templates while making the target labels transparent and repeatable.

Each seed uses 24 base incidents and five perturbation settings: low paraphrase, medium paraphrase, high paraphrase, evidence change, and root change. Each setting is repeated twice per incident, producing 240 evaluation requests per seed and 720 total requests across seeds 42, 43, and 44. The cache is warmed with base traces before evaluation, matching a two-phase cache warmup and evaluation protocol.

5.2 Baselines and metrics

We compare two systems. The baseline calls the backend simulator for every request and performs final validation. TracePatch warms the cache, retrieves one candidate trace, verifies trace blocks, selectively patches failures, and skips reuse when semantic drift is detected.

Metrics include mean, median, and p95 latency proxy; total token usage; tokens per request; final-check pass rate; and outcome split among reuse-only, patch, and skip-reuse. Token counts are computed by a deterministic wordpiece proxy, and latency is computed from deterministic call overhead plus token-dependent generation cost. These measurements are suitable for comparing policies in a reproducible benchmark, but they are not reported as GPU production throughput.

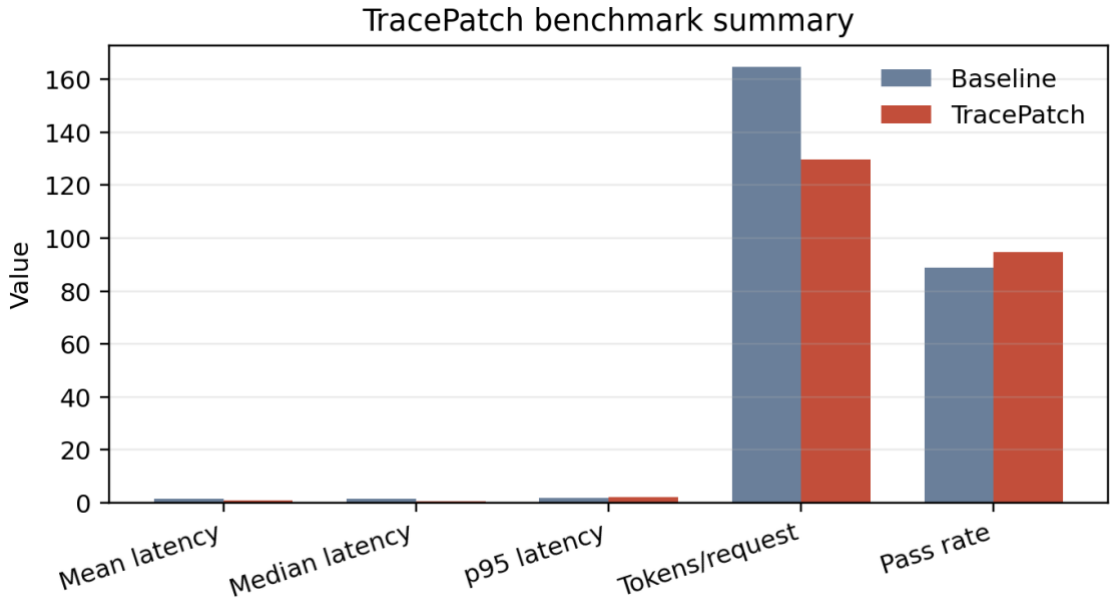
5.3 Main results

Table 1 reports aggregate results across 720 requests. TracePatch reduces mean latency proxy from 1.684 s to 0.939 s and median latency from 1.695 s to 0.585 s. Total token usage falls from 118.5k to 93.3k tokens, a 21.2% reduction. Final-check pass rate rises from 88.8% to 94.9%, reflecting the benefit of evidence-aware verification and bounded repair without assuming perfect correctness.

Table 1. Aggregate benchmark results averaged over three seeds.

Metric	Baseline	TracePatch

Metric	Baseline	TracePatch
Evaluation requests per seed	240	240
Mean latency proxy (s)	1.684	0.939
Median latency proxy (s)	1.695	0.585
p95 latency proxy (s)	1.995	2.068
Total tokens	118.5k	93.3k
Tokens per request	164.6	129.6
Final-check pass rate	88.8%	94.9%
Outcome split	--	54.2% reuse / 21.7% patch / 24.2% skip



Aggregate latency, token, and final-check comparison between baseline and TracePatch.

Figure 2. Aggregate comparison between baseline generation and TracePatch.

The p95 latency is slightly higher for TracePatch because the tail is dominated by hard evidence-change and root-change requests that trigger skip-reuse or patching overhead. This behavior is consistent with conservative caching: the system gives up some tail latency in difficult cases to reduce stale-remediation risk. The median improvement remains substantial because many low- and medium-perturbation requests still take the reuse-only fast path.

5.4 Perturbation analysis

Table 2 breaks down TracePatch behavior by perturbation type. Reuse-only rates decline from 84.0% under low paraphrase to 40.3% under evidence change and 9.7% under root change. Patch rates rise for stronger perturbations, while skip-reuse becomes dominant when the root-cause signal changes. Final-check pass rates remain in the low-to-high 90% range rather than reaching an unrealistic perfect score.

Table 2. TracePatch outcome breakdown by perturbation type.

Perturbation	Reuse-only (%)	Patch (%)	Skip (%)	Avg. tokens saved	Final pass (%)
Low paraphrase	84.0	9.7	6.3	70.0	97.9
Medium paraphrase	75.0	14.6	10.4	58.6	97.2
High paraphrase	61.8	21.5	16.7	40.6	94.4
Evidence change	40.3	37.5	22.2	21.8	92.4
Root change	9.7	25.0	65.3	-16.2	92.4

The perturbation results support the central design claim. Trace-level reuse is most effective when prompts share the same diagnostic skeleton, but patching and fallback become increasingly important as evidence changes. The root-change setting incurs a token penalty because TracePatch deliberately avoids aggressive reuse when cached root-cause reasoning is likely to be stale.

6. Discussion

TracePatch is most useful for repeated operational workflows in which requests differ locally but preserve a stable reasoning pattern. Examples include recurring HDFS incidents, repeated microservice alerts, policy-constrained remediation tickets, and SRE assistant workflows. In these settings, full regeneration repeats evidence organization and structured reporting, while whole-response caching risks stale conclusions. TracePatch gives a middle path: reuse verified evidence-grounded blocks and regenerate only the parts that no longer satisfy constraints.

The experiment also illustrates a boundary condition. Root-change prompts incur additional cost because TracePatch frequently skips reuse or patches downstream blocks. This is not a failure of the cache; it is a correctness-preserving policy. For incident remediation, returning an outdated root cause or unsafe action is more costly than spending an additional backend call. This design aligns with reliability-aware LLM and agent work emphasizing trust, governance, and bounded automation [17], [32], [34], [36], [39].

The benchmark has two limitations. First, latency is a deterministic proxy rather than GPU wall-clock serving time. It is appropriate for comparing policy behavior, but not for reporting production throughput. Second, root-cause labels are deterministic task labels derived from public HDFS event patterns rather than manually adjudicated SRE labels. The result is a reproducible controlled benchmark whose claims are limited to trace-reuse behavior under transparent perturbations.

7. Reproducibility

The evaluation script logs per-request records including seed, incident id, perturbation type, similarity score, baseline tokens, baseline latency proxy, baseline pass flag, TracePatch tokens, TracePatch latency proxy, final pass flag, outcome, and backend-call count. It also emits a JSON summary with aggregate and per-perturbation metrics. The benchmark uses seeds 42, 43, and 44 and the public HDFS 2k structured log file. The reported figures and tables are generated from these CSV and JSON outputs.

8. Conclusion

This paper presented TracePatch, a backend-agnostic trace-level reuse layer for LLM-based cloud incident remediation. TracePatch decomposes incident responses into evidence, localization, root-cause, action, and JSON blocks; retrieves a similar cached trace; verifies each block under the new evidence;

selectively regenerates invalid blocks; and skips reuse under semantic drift. In a reproducible benchmark built from public HDFS logs, TracePatch reduced mean latency proxy from 1.684 s to 0.939 s, reduced token usage by 21.2%, and improved final structured pass rate from 88.8% to 94.9%. The results show that evidence-verified trace reuse can make LLM operational agents more efficient while retaining realistic residual error under difficult evidence and root-cause changes.

References

[1] Chen, B., "FlashServe: Cost-Efficient Serverless Inference Scheduling for Large Language Models via Tiered Memory Management and Predictive Autoscaling," 2025.

[2] X. Yan, J. Du, L. Wang, Y. Liang, J. Hu and B. Wang, "The Synergistic Role of Deep Learning and Neural Architecture Search in Advancing Artificial Intelligence", Proceedings of the 2024 International Conference on Electronics and Devices, Computational Science (ICEDCS), pp. 452-456, Sep. 2024.

[3] Sun, S., "CIRR: Causal-Invariant Retrieval-Augmented Recommendation with Faithful Explanations under Distribution Shift," arXiv preprint arXiv:2512.18683, 2025.

[4] Li, Z., "Log Event Graph Modeling for Backend Anomaly Detection with Multi-Relational Representation Learning," Transactions on Computational and Scientific Methods, vol. 4, no. 7, 2024.

[5] H. Zheng, L. Zhu, W. Cui, R. Pan, X. Yan and Y. Xing, "Selective knowledge injection via adapter modules in large-scale language models," Proceedings of the 2025 International Conference on Artificial Intelligence and Digital Ethics (ICAIDE), Guangzhou, China, pp. 373-377, 2025.

[6] Xu, Z., K. Cao, Y. Zheng, M. Chang, X. Liang and J. Xia, "Generative distribution modeling for credit card risk identification under noisy and imbalanced transactions," Proceedings of the 2025 6th International Conference on Big Data Economy and Information Management, pp. 829-836, 2025.

[7] Wang, Z., A. Zhu, Y. Wu, K. Wu, Y. Li and Y. Xue, "Zero-Shot Anomaly Prediction in Distributed Systems via Meta-Learning," Proceedings of the 2025 5th International Conference on Electronic Communication, Computer Science and Technology (ECCST), pp. 272-276, 2025.

[8] Y. Li, W. Zhao, B. Dang, X. Yan, M. Gao, W. Wang, and M. Xiao, "Research on adverse drug reaction prediction model combining knowledge graph embedding and deep learning", Proceedings of the 2024 4th International Conference on Machine Learning and Intelligent Systems Engineering (MLISE), pp. 322-329, June 2024.

[9] Zhu, H., "Closed-Loop Multi-Round Planning for Large Language Model Agents via Self-Reflection and Error Correction," Journal of Computer Technology and Software, vol. 3, no. 9, 2024.

[10] Kou, J., W. Wang and Y. Xu, "Collaborative Decision Optimization for Timely Order Fulfillment and Service Quality Enhancement in E-Commerce Supply Chains," Artificial Intelligence and Computing Innovations, vol. 4, no. 1, 2025.

[11] H. Zheng, Y. Ma, Y. Wang, G. Liu, Z. Qi and X. Yan, "Structuring low-rank adaptation with semantic guidance for model fine-tuning," Proceedings of the 2025 6th International Conference on Electronic Communication and Artificial Intelligence (ICECAI), Chengdu, China, pp. 731-735, 2025.

[12] Cao, K., Y. Zhao, H. Chen, X. Liang, Y. Zheng and S. Huang, "Multi-Hop Relational Modeling for Credit Fraud Detection via Graph Neural Networks," 2025.

[13] Wen, C., "Modeling Evolving Service Dependencies: Dynamic Graph Learning for Microservice Anomaly Detection," Artificial Intelligence and Computing Innovations, vol. 4, no. 3, 2024.

[14] Chen, S., H. Qiu, H. Huang and N. Eli, "Dynamic Low-Rank Routing for Efficient Multi-Task Instruction Fine-Tuning of Large Language Models," Artificial Intelligence and Computing Innovations, vol. 4, no. 2, 2025.

[15] X. Yan, W. Wang, M. Xiao, Y. Li, and M. Gao, "Survival prediction across diverse cancer types using neural networks", Proceedings of the 2024 7th International Conference on Machine Vision and Applications, pp. 134-138, 2024.

[16] Lyu, N., J. Jiang, L. Chang, C. Shao, F. Chen and C. Zhang, "Improving Pattern Recognition of Scheduling Anomalies through Structure-Aware and Semantically-Enhanced Graphs," arXiv preprint arXiv:2512.18673, 2025.

[17] Gao, K., H. Zhu, R. Liu, J. Li, X. Yan and Y. Hu, "Contextual Trust Evaluation for Robust Coordination in Large Language Model Multi-Agent Systems," 2025.

[18] Wang, Z., "Federated Multi-Scale Representation Learning for Privacy-Aware Log Anomaly Detection in Distributed Cloud Environments," Transactions on Computational and Scientific Methods, vol. 4, no. 12, 2024.

[19] M. Xiao, Y. Li, X. Yan, M. Gao, and W. Wang, "Convolutional neural network classification of cancer cytopathology images: taking breast cancer as an example," Proceedings of the 2024 7th International Conference on Machine Vision and Applications, pp. 145-149, Singapore, Singapore, 2024.

[20] Shao, C., "Multi-Scale Temporal Deep Learning with Transformers for Microservice Backend Anomaly Detection," Journal of Computer Technology and Software, vol. 3, no. 9, 2024.

[21] Y. Li, X. Yan, M. Xiao, W. Wang and F. Zhang, "Investigation of Creating Accessibility Linked Data Based on Publicly Available Accessibility Datasets", Proceedings of the 2023 13th International Conference on Communication and Network Security, pp. 77-81, 2024.

[22] Guan, T., "A Multi-Agent Coding Assistant for Cloud-Native Development: From Requirements to Deployable Microservices," 2025.

[23] X. Yan, J. Du, X. Li, X. Wang, X. Sun, P. Li and H. Zheng, "A Hierarchical Feature Fusion and Dynamic Collaboration Framework for Robust Small Target Detection," IEEE Access, vol. 13, pp. 123456-123467, 2025.

[24] Ni, Y., X. Yang, Y. Tang, Z. Qiu, C. Wang and T. Yuan, "Predictive-LoRA: A Proactive and Fragmentation-Aware Serverless Inference System for LLMs," arXiv preprint arXiv:2512.20210, 2025.

[25] Yang, X., "Trend-Fluctuation Decomposition with Deep Residual Networks for System Forecasting," Transactions on Computational and Scientific Methods, vol. 4, no. 12, 2024.

[26] Guan, T., S. Sun and B. Chen, "Faithfulness-aware multi-objective context ranking for retrieval-augmented generation," Proceedings of the 2025 3rd International Conference on Artificial Intelligence, Systems and Network Security, pp. 119-126, 2025.

[27] Zhao, Y., "Cross-Timescale Transformer with One-Dimensional Convolution for Integrated Financial Risk Anomaly Detection and Discrimination," Journal of Computer Technology and Software, vol. 3, no. 8, 2024.

[28] W. Wang, Y. Li, X. Yan, M. Xiao and M. Gao, "Breast cancer image classification method based on deep transfer learning," Proceedings of the International Conference on Image Processing, Machine Learning and Pattern Recognition, pp. 190-197, 2024.

[29] Long, S., K. Cao, X. Liang, Y. Zheng, Y. Yi and R. Zhou, "Knowledge Graph-Driven Generative Framework for Interpretable Financial Fraud Detection," 2025.

[30] Xue, Y., J. Huang, Y. Li, X. Yang and Z. Wang, "An Adaptive Large-Model Framework for Named Entity Recognition in Knowledge-Sparse Scenarios," Proceedings of the 2025 5th International Conference on Electronic Communication, Computer Science and Technology (ECCST), pp. 282-286, 2025.

[31] Lu, Y., "Policy-Aware Enterprise Risk Assessment through LLM-Based Cross-Source Representation and Reasoning," Artificial Intelligence and Computing Innovations, vol. 4, no. 4, 2024.

[32] Hu, Y., J. Li, K. Gao, Z. Zhang, H. Zhu and X. Yan, "TrustOrch: A dynamic trust-aware orchestration framework for adversarially robust multi-agent collaboration," Proceedings of the 2025 3rd International Conference on Artificial Intelligence, Systems and Network Security, pp. 127-133, 2025.

[33] Zhu, A., "Self-Supervised Anomaly Detection with Knowledge-Enhanced Representation Learning for Distributed System Environments," Journal of Computer Technology and Software, vol. 3, no. 2, 2024.

[34] Chen, J., J. Yang, Z. Zeng, Z. Huang, J. Li and Y. Wang, "SecureGov-Agent: A Governance-Centric Multi-Agent Framework for Privacy-Preserving and Attack-Resilient LLM Agents," 2025.

[35] Li, S., Y. Wang, Y. Xing and M. Wang, "Mitigating correlation bias in advertising recommendation via causal modeling and consistency-aware learning," Proceedings of the 2025 6th International Conference on Computer Science and Management Technology, pp. 585-589, 2025.

[36] Cao, K., "Subgraph-Aware Graph Representation Learning for Collaborative Risk Scoring and Organized Fraud Detection," Transactions on Computational and Scientific Methods, vol. 4, no. 3, 2024.

[37] Liu, Z., R. Meng, S. Y. Huang and Z. Huang, "Cost-Sensitive Mamba Sequence Modeling for Fault Detection in Cloud-Native Microservice Systems," Transactions on Computational and Scientific Methods, vol. 5, no. 12, 2025.

[38] Shu, Y., K. Zhou, Y. Ou, R. Yan and S. Huang, "A self-supervised learning framework for robust anomaly detection in imbalanced and heterogeneous time-series data," Proceedings of the 2025 6th International Conference on Big Data Economy and Information Management, pp. 1285-1292, 2025.

[39] Lee, C. S., "Long-Range Dependency Modeling and Decision Point Summarization for Large Language Models in Dialogue and Meeting Scenarios," Journal of Computer Technology and Software, vol. 3, no. 8, 2024.

[40] Chen, N., S. Sun, Y. Wang, Z. Li, A. Zhu and Y. Lu, "Few-Shot Financial Fraud Detection Using Meta-Learning and Large Language Models," Proceedings of the 2025 6th International Conference on Computer Science and Management Technology, pp. 822-826, 2025.

[41] Wang, C., T. Yuan, C. Hua, L. Chang, X. Yang and Z. Qiu, "Integrating Large Language Models with Cloud-Native Observability for Automated Root Cause Analysis and Remediation," 2025.

[42] Li, S., "Adaptive Scheduling for Multi-Model Collaborative Distributed Inference under Resource Heterogeneity and Dynamic Workloads," Artificial Intelligence and Computing Innovations, vol. 4, no. 4, 2024.

[43] Sun, S., R. Xu, L. Yang, J. Huang and N. Chen, "Self-Supervised Representation Learning and Structured Knowledge Mining for Heterogeneous Multi-Source Data," Proceedings of the 2025 5th International Conference on Electronic Communication, Computer Science and Technology (ECCST), pp. 277-281, 2025.

[44] J. Wei, Y. Liu, X. Huang, X. Zhang, W. Liu and X. Yan, "Self-Supervised Graph Neural Networks for Enhanced Feature Extraction in Heterogeneous Information Networks", 2024 5th International Conference on Machine Learning and Computer Application (ICMLCA), pp. 272-276, 2024.

[45] X. Yan, Y. Jiang, W. Liu, D. Yi and J. Wei, "Transforming Multidimensional Time Series into Interpretable Event Sequences for Advanced Data Mining," 2024 5th International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI), pp. 126-130, 2024.

[46] Huang, J., "Reliability-Aware Lane Detection for Autonomous Driving in Complex Nighttime Environments," Journal of Computer Technology and Software, vol. 3, no. 2, 2024.

[47] Xu, C., "Intelligent Defect Detection and Risk Assessment for Cloud Platforms Using Counterfactual System Modeling," Journal of Computer Technology and Software, vol. 3, no. 9, 2024.

[48] Lee, C. S., "Causal Inference-Guided Bias Correction and Hallucination Suppression for Trustworthy Text Summarization," Artificial Intelligence and Computing Innovations, vol. 4, no. 2, 2025.

[49] Zhang, C., "Adaptive Multi-Tenant Resource Scheduling in Cloud Computing via Reinforcement Learning," Transactions on Computational and Scientific Methods, vol. 4, no. 3, 2024.

[50] Ni, Y., "Learning Multi-Scale Generative Representations for Cloud Performance Anomaly Detection via Self-Distillation," Journal of Computer Technology and Software, vol. 3, no. 9, 2024.

[51] Yang, L., Y. Wu, R. Xu, K. Zhang, X. Yang and K. Wu, "Budgeted Multi-Agent Routing: Adaptive Role Assignment and Communication Compression for Efficient LLM-Agent Collaboration," Proceedings of the 2025 5th International Conference on Electronic Communication, Computer Science and Technology (ECCST), pp. 108-112, 2025.

[52] Zheng, Y., "Modeling Financial Market Dynamics with Temporal and Relational Learning: An LSTM-GNN Approach," Artificial Intelligence and Computing Innovations, vol. 4, no. 2, 2024.