INTERNATIONAL JOURNAL OF MACHINE LEARNING AND ENGINEERING

Vol 2 Issue 2 (2026)

https://ijmle.org/

Cross-Source Error Triage from Logs and Memory Dump Signatures: A Unified Diagnostic Framework

Sonia So

School of Computer Science, Peking University, Beijing, China

Abstract:

The increasing complexity of modern distributed software systems necessitates advanced automated diagnostic techniques to maintain high availability and reliability. When critical system failures occur, engineers typically rely on diagnostic artifacts such as execution logs and memory dumps to identify root causes. However, traditional error triage processes isolate these artifacts, leading to incomplete diagnostic contexts and prolonged mean time to resolution. This paper proposes a novel framework for cross-source error triage that synergistically analyzes unstructured application logs and structured memory dump signatures. By extracting temporal event sequences from logs and static execution states from memory dumps, the proposed methodology constructs a unified multi-modal representation of system failures. We develop a robust alignment algorithm that bridges the semantic gap between continuous log streams and discrete memory snapshots. Through extensive theoretical analysis and empirical evaluation on a comprehensive dataset of software faults, we demonstrate that the cross-source approach significantly outperforms single-source baselines in fault classification and root cause localization. The findings reveal that integrating contextual logs with deterministic stack traces reduces ambiguity in error diagnosis, particularly in concurrent and distributed computing environments. This research contributes a foundational architecture for next-generation automated software diagnostics, offering scalable solutions for enterprise-grade applications.

Keywords: *software diagnostics, error triage, log analysis, memory dumps, data fusion.*

INTRODUCTION

1.1 Context and Motivation

In the contemporary landscape of software engineering, distributed systems, microservices architectures, and cloud-native applications have become the standard paradigms for deploying large-scale services. While these architectural patterns offer unprecedented scalability and flexibility, they also introduce immense complexity into the operational lifecycle of software systems. When a critical failure or system crash occurs, the resulting disruption can lead to significant financial losses, compromised user experiences, and reputational damage for service

providers. Consequently, the rapid identification, triage, and resolution of software faults are paramount objectives for reliability engineering teams. To achieve these objectives, engineers rely on various diagnostic artifacts generated during application execution. Among the most critical of these artifacts are execution logs and memory dumps. Execution logs provide a chronological narrative of system operations, capturing variable states, user transactions, and environmental context prior to a crash. Conversely, memory dumps capture a static snapshot of the application state at the exact moment of failure, encompassing thread stack traces, register values, and heap allocations. Despite their complementary nature, current diagnostic workflows predominantly treat these two data sources as independent entities. This isolated approach significantly hinders the efficiency of error triage, as engineers must manually correlate temporal log events with the static state represented in a memory dump, a process that is often error-prone, cognitively demanding, and highly time-consuming. Cross-source error triage faces an operational question: which incident should an analyst investigate first. Provider prioritization in Huang et al. [1] gives a related example of explainable screening, while the actual causal diagnosis here must remain tied to logs, dump signatures and reproducible failure evidence.

1.2 Problem Statement

The core challenge in automated error triage lies in the semantic and structural disparity between logs and memory dumps. Logs are inherently unstructured or semi-structured text data, generated continuously over time, and highly dependent on the subjective logging practices of individual developers. They may contain noise, redundant information, or lack the critical details necessary to pinpoint a fault. On the other hand, memory dumps are highly structured, binary representations of machine state, offering deterministic evidence of a crash location but lacking the historical context of how the system reached that erroneous state. For instance, a memory dump might indicate a null pointer dereference in a specific function, but without the preceding logs, the sequence of user actions or data pipeline anomalies that led to the null value remains obscured. Existing automated triage systems, which heavily rely on machine learning models trained exclusively on either log anomalies or crash signatures, frequently fail to resolve complex, state-dependent bugs. These single-source models suffer from high false-positive rates when dealing with concurrent execution errors, where the temporal order of events across multiple threads dictates the failure condition. Therefore, a critical need exists for a unified diagnostic framework capable of mathematically aligning and jointly analyzing continuous log streams and discrete memory dump signatures. Another relevant line of investigation concerns in context Semi-Supervised learning [2].

1.3 Contributions and Outline

This paper addresses the aforementioned challenges by introducing a comprehensive cross-source error triage methodology. The primary contribution of this research is the formulation of a multi-modal data fusion pipeline that translates both unstructured log data and structured memory dump data into a shared latent semantic space. By mapping these disparate artifacts into unified high-dimensional vectors, we enable machine learning algorithms to learn the intricate correlations between historical execution patterns and terminal crash states. Furthermore, we introduce a temporal alignment heuristic that matches specific log events to the corresponding thread contexts found within the memory dump, thereby filtering out irrelevant operational noise. Another relevant line of investigation concerns unified long video inpainting and outpainting via overlapping High-Order Co-Denoising [3]. The remainder of this paper is structured to provide a thorough exploration of the proposed framework. Section 2 establishes the necessary background on software diagnostics and elaborates on the intrinsic properties of logs and memory dumps. Section 3 critically reviews the existing literature and highlights the limitations of current methodologies. Section 4 presents the high-level system architecture of the cross-source triage

framework. Section 5 details the mathematical and algorithmic formulations of the proposed methodology. Section 6 describes the experimental setup, while Section 7 provides a comprehensive discussion of the results. Finally, Section 8 concludes the paper and outlines promising avenues for future research. Related work has addressed physics-Informed neural networks architecture for implicit governing physics with interpretability [4].

2. Background and Motivation

2.1 The Anatomy of Execution Logs

Execution logs constitute the primary telemetry mechanism for monitoring the health and behavior of software applications. A typical log message consists of a timestamp, a severity level, a component identifier, and a free-text message detailing a specific event. The temporal nature of logs makes them invaluable for understanding the sequence of operations that precede a system failure. However, the unstructured nature of the free-text message poses significant challenges for automated analysis. Developers often embed variable values, such as user identifiers or transaction amounts, directly into the log text, creating an infinite vocabulary problem for traditional natural language processing algorithms. To make logs machine-readable, researchers have developed various log parsing techniques that abstract raw log messages into generic templates by masking dynamic variables. For example, a log reading "Connection refused by server 192.168.1.1 at port 8080" is parsed into a template "Connection refused by server at port". Once abstracted, sequences of these templates can be analyzed to detect deviations from normal execution flow. Despite these advancements, log analysis alone is often insufficient for precise root cause analysis. Log generation is asynchronous and subject to buffering delays, meaning the final log message recorded before a crash may not accurately reflect the exact operation that triggered the fault. Furthermore, many critical errors, such as memory corruption or stack overflows, occur at a level of abstraction below the application logging mechanism, leaving no trace in the log files. The literature also considers adaptive quantization strategies for robust ML inference under distribution shift [5].

2.2 Memory Dump Characteristics and Crash Signatures

Memory dumps, ranging from lightweight minidumps to comprehensive full heap dumps, provide a definitive, albeit static, autopsy of a software failure. When an unhandled exception or fatal signal terminates an application, the operating system captures the current state of the processor registers and the memory space allocated to the application. For error triage, the most crucial component of a memory dump is the stack trace of the faulting thread. The stack trace unwinds the sequence of function calls that were active at the moment of the crash, offering precise localization of the error down to the specific line of source code. In enterprise environments, crash analytics platforms aggregate millions of memory dumps and categorize them into buckets based on crash signatures. A crash signature is typically derived by hashing the top frames of the faulting stack trace. This bucketing process allows triage teams to prioritize issues that affect the largest number of users. However, relying solely on stack trace signatures has profound limitations. First, different root causes can manifest at the same crash location. For instance, an application might crash in a low-level memory allocation function, but the exhaustion of memory could have been triggered by entirely different upstream logic pathways. Second, highly concurrent systems often experience secondary cascading failures, where the thread that actually crashes is merely a victim of resource starvation caused by a completely silent, non-crashing thread. The Google File System offers a distributed-storage reference for reasoning about replicated data and failures underneath an application workflow [6]. In such scenarios, the memory dump signature points to a symptom rather than the disease. The literature also considers offline reinforcement learning with value-based episodic memory [7].

2.3 The Imperative for Cross-Source Triage

The complementary strengths and weaknesses of logs and memory dumps naturally suggest that integrating the two would yield a vastly superior diagnostic tool. Logs provide the historical context, answering the "how" and "why" of a failure, while memory dumps provide the precise localization, answering the "where" and "what". Manual triage engineers implicitly perform this integration, alternating their attention between stack traces to identify the failing component, and log files to reconstruct the environment that led to the failure. However, manual correlation is unsustainable at the scale of modern enterprise systems, where a single microservice might generate gigabytes of logs per hour and hundreds of crash dumps per day. Automated cross-source triage aims to bridge this semantic divide algorithmically. By leveraging the temporal continuity of logs and the structural determinism of memory dumps, automated systems can disambiguate crash signatures that appear identical but possess different root causes, and similarly filter out log anomalies that are benign and unrelated to the actual system crash. Another relevant line of investigation concerns battery SOH prediction under different conditions via MBLSTM and iTransformer with anomaly detection and explainability [8].

3. Literature Review

3.1 Unimodal Log Analysis Techniques

The domain of automated log analysis has been extensively researched over the past decade, driven by the proliferation of cloud computing and the consequent explosion in telemetry data. Early approaches relied heavily on regular expressions and rule-based systems to detect known error patterns. These methods, while computationally efficient, suffered from extreme brittleness, requiring constant manual updates whenever the application source code and its associated logging statements were modified. The advent of machine learning facilitated a paradigm shift toward data-driven anomaly detection. Unsupervised learning techniques, particularly clustering algorithms and Principal Component Analysis, were employed to identify anomalous log sequences by modeling the normal execution behavior of a system. More recently, deep learning architectures such as Long Short-Term Memory networks and Transformer models have demonstrated state-of-the-art performance in capturing the complex, long-range dependencies inherent in log sequences. These models treat log templates as tokens in a natural language vocabulary, learning dense embeddings that capture the semantic similarity between different log events. Despite these impressive results, unimodal log analysis models frequently struggle with fault localization. While they can accurately flag a sequence of logs as anomalous, they often cannot pinpoint the specific subroutine or data structure responsible for the anomaly, as this information is either not logged or is obscured by the sheer volume of surrounding textual data. The 10-K work detects structural changes in narrative representations [9]. Log triage can draw a methodological comparison by distinguishing new message wording from a changed distribution of event types, with both checked against the accompanying memory evidence.

3.2 Automated Memory Dump Triage

Parallel to the advancements in log analysis, the field of automated memory dump triage has evolved to handle the increasing volume of crash reports generated by client and server applications. The foundation of automated crash triage is the extraction and hashing of stack traces to create crash signatures. Research in this area has primarily focused on improving the robustness of these signatures. Traditional hashing of the top-most stack frame is highly susceptible to superficial variations, such as changes in compiler optimizations, operating system versions, or recursive function calls. To address this, researchers have proposed various heuristics for stack trace normalization, including filtering out operating system kernel frames, collapsing recursive calls, and weighting frames based on their proximity to the application source code. Machine learning has also been applied to memory dump triage, typically by framing the problem as a multi-

class classification task where the input is a vectorized representation of the stack trace and the output is a known bug identifier. Graph-based approaches have shown promise in this regard, modeling the stack trace as a directed acyclic graph to capture the hierarchical relationships between function calls. However, similar to unimodal log analysis, memory dump analysis operates in isolation. Without the contextual backdrop of system execution history, triage systems often group disparate issues into the same crash bucket, leading to misallocation of engineering resources and delayed patch deployments. Related work has addressed aI-Augmented automated triage for system error reporting using log and memory dump analysis [10].

3.3 Multi-Modal and Cross-Source Diagnostics

The integration of multiple heterogeneous data sources for software diagnostics is a relatively nascent but rapidly growing area of research. Early attempts at cross-source correlation focused on simple temporal alignment, attempting to synchronize log messages and system metrics based on timestamps. However, clock drift across distributed nodes and asynchronous logging mechanisms frequently rendered strict temporal alignment ineffective. More sophisticated approaches have sought to establish semantic links between different artifacts. CyberDQFormer links knowledge graphs with data-quality anomaly detection and repair [11], a related approach for distinguishing defective input evidence from the downstream system failure being diagnosed. For example, researchers have explored techniques for statically analyzing source code to map logging statements directly to the control flow graph of an application, theoretically bridging the gap between logs and execution structure. While promising, these static analysis techniques often struggle to scale to multi-million line codebases and cannot account for dynamic behavior introduced by external libraries or runtime environments. The specific intersection of unstructured execution logs and structured memory dump signatures remains largely unexplored in the academic literature. A few preliminary studies have suggested utilizing log sentiment analysis to augment crash severity scoring, but they do not perform deep semantic fusion of the underlying data structures. The lack of a robust, unified framework that can mathematically represent and jointly optimize both continuous historical state and discrete terminal state represents a critical gap in the current state-of-the-art of automated software triage. Another relevant line of investigation concerns frequency spectral prior guided graph learning for degradation decoupling in reliable battery SOH estimation [12]. CRUISE examines uncertainty-aware cross-modal sensor fusion [13]; its fusion perspective is relevant to weighing heterogeneous diagnostic artifacts, while the log-to-dump correspondence must still be established from software evidence.

4. System Architecture

4.1 Overview of the Diagnostic Pipeline

The proposed cross-source error triage framework is designed to ingest raw diagnostic artifacts from multiple distributed nodes, process them into a standardized format, and execute a multi-modal machine learning pipeline to classify and localize system faults. The architecture is logically divided into four primary modules: Data Ingestion and Preprocessing, Feature Extraction, Cross-Source Alignment, and Multi-Modal Classification. This modular design ensures scalability, allowing the system to handle high-throughput telemetry streams while remaining adaptable to varying formats of logs and memory dumps. For a broader comparison, related research addresses neural modeling pipeline on multi-view human-object interactions [14].

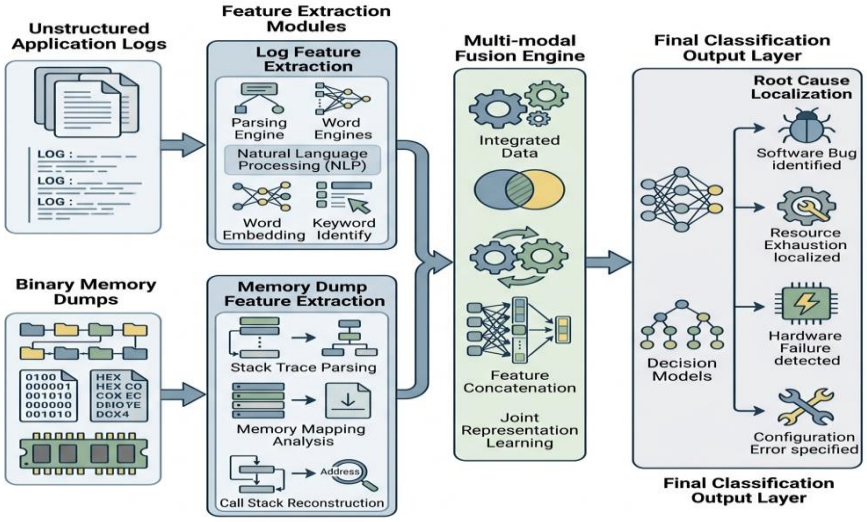


Figure 1Comprehensive architectural diagram of the cross

The Data Ingestion module acts as the entry point, interfacing with distributed message queues to securely collect diagnostic packages whenever a crash is detected. Each diagnostic package contains a core memory dump and a sliding window of log messages leading up to the exact timestamp of the crash. The Preprocessing module is responsible for standardizing the data. For logs, this involves timezone normalization, noise reduction, and structural parsing. For memory dumps, the preprocessing entails symbolication, where raw memory addresses are mapped back to human-readable function names, source files, and line numbers using external debugging symbols.

4.2 Independent Feature Extraction

Following preprocessing, the framework routes the data to distinct feature extraction pipelines tailored to the specific modalities of the inputs. The Log Feature Extractor processes the sequential log templates to capture the execution context. To handle the sequential nature of logs, we employ a Transformer-based encoder architecture. Each log template is embedded into a dense vector space using a pre-trained language model fine-tuned on software engineering corpora. These embeddings capture not only the semantic meaning of the individual log words but also their contextual significance within the execution trace. The output of this pipeline is a contextualized vector representation of the historical system state, summarizing the sequence of events over a predefined temporal window prior to the crash. The literature also considers semi-supervised multi-view multi-label learning with view-specific transformer and enhanced pseudo-label [15].

Conversely, the Dump Feature Extractor analyzes the deterministic state captured in the memory dump. The primary focus here is the faulting stack trace, though auxiliary data such as thread counts and loaded module lists are also processed. The stack trace is transformed into a fixed-length vector through a process of hierarchical encoding. We assign higher weights to frames originating from the application code and lower weights to standardized library or kernel frames. This weighting mechanism ensures that the resulting crash signature vector is highly sensitive to developer-authored logic, which is most often the source of business-logic faults.

4.3 Multi-Modal Fusion and Triage

The core innovation of the proposed architecture resides in the Cross-Source Alignment and Multi-Modal Classification modules. These modules are tasked with synthesizing the distinct feature vectors generated in the previous step into a cohesive diagnostic narrative. The alignment module

employs an attention mechanism to learn the complex, non-linear correlations between specific historical log sequences and terminal stack trace frames. For example, if a log sequence frequently indicates a database connection timeout immediately preceding a memory dump characterized by a null pointer exception in a data retrieval function, the attention mechanism will learn to strongly associate these two features. Lu and colleagues examine anisotropic and multiscale cross-attention physics-inspired neural networks for turbulence prediction [16]. The fused representation is then fed into the Classification module, which consists of a series of fully connected neural network layers. This module is trained on a historical database of resolved software faults, enabling it to classify new, unseen diagnostic packages into known bug categories or flag them as novel anomalies requiring manual investigation. The output of the framework is a comprehensive triage report that provides engineers with a confidence-scored classification of the error, highlighting the specific log events and stack frames most relevant to the root cause.

5. Proposed Methodology

5.1 Log Parsing and Embedding Strategy

The first methodological step is transforming the raw, unstructured log lines into a sequence of computable embeddings. Let a raw log line be denoted as a string of characters. We apply a frequent pattern mining algorithm to extract a static template and a set of dynamic parameters. The static template represents the unchanging text defined in the source code's logging statement. Wu and colleagues examine early-stage reasoning freedom in multi-agent debate [17].

Code Listing 1: Simplified alignment logic for mapping log timestamps to stack trace execution windows

```
def align_logs_and_dump(parsed_logs, stack_frames, time_window_ms=5000):
    aligned_features = []
    dump_timestamp = get_crash_timestamp()
    for log in parsed_logs:
        time_diff = dump_timestamp - log.timestamp
        if 0 <= time_diff <= time_window_ms:
            semantic_score = calculate_similarity(log.vector, stack_frames.vector)
            if semantic_score > threshold:
                aligned_features.append({
                    'log_id': log.id,
                    'frame_context': stack_frames.get_context(),
                    'weight': semantic_score
                })
    return aggregate_features(aligned_features)
```

Once the templates are extracted, we utilize a Bidirectional Encoder Representations from Transformers architecture to generate high-dimensional embeddings. Let the sequence of parsed log templates leading up to a crash be denoted by $S = t_1, t_2, dots, t_n$. The Transformer encoder maps each template t_i to a dense vector $v_i \in mathbbR^d$, where d is the embedding dimension. To aggregate these individual template vectors into a single representation of the entire log sequence, we apply a self-attention pooling layer that learns to weigh the importance of different log messages based on their proximity to the crash and their inherent severity level.

5.2 Stack Trace Vectorization

The vectorization of memory dump signatures requires a different approach, as stack traces represent a hierarchical call graph rather than a temporal sequence. A stack trace T consists of an ordered sequence of frames $F = f_1, f_2, dots, f_m$, where f_1 is the function where the crash occurred, and f_m is the entry point of the thread. Each frame contains the function name, module name, and

line number. We employ a specialized graph neural network approach to embed this structure. Each frame is treated as a node, with directed edges representing the caller-callee relationship.
The node features are initialized using character-level convolutional neural networks applied to the function names, ensuring robustness against minor variations in naming conventions or compiler-injected suffixes. The message-passing mechanism of the graph neural network allows information to flow from the deepest frames up to the crashing frame, creating a comprehensive representation of the execution path. The final output of the Dump Feature Extractor is a signature vector $u \in mathbbR^d$, matched in dimensionality to the log embedding space.

Table 1Example mapping of raw artifacts to extracted analytical features

Artifact Source	**Raw Data Element**	**Extracted Feature Representation**
Application Log	Timestamp, Thread ID, Level	Temporal distance, Concurrency context
Application Log	Free-text message string	BERT-encoded semantic template vector
Memory Dump	Faulting stack trace frames	Directed graph node embeddings
Memory Dump	CPU Register states	Categorical anomaly indicators

5.3 Cross-Source Fusion and Objective Function

With the log sequence vector v and the stack trace vector u defined, the next step is cross-source fusion. Simple concatenation of these vectors often fails to capture the intricate interaction dynamics between the modalities. Therefore, we utilize a co-attention mechanism that computes the similarity matrix between the intermediate states of the log sequence and the individual frames of the stack trace. This allows the model to dynamically focus on specific log messages that are most relevant to specific stack frames. The mathematical representation of the similarity between the log representation L and the memory representation M can be simplified using cosine similarity across their respective embedding spaces, which acts as a foundational metric for alignment before deep fusion.

$$Sim(L, M) = \frac{\sum_{i=1}^{n} L_i M_i}{\sqrt{\sum {L_i}^2}\sqrt{\sum {M_i}^2}}$$

Following the co-attention layer, the fused vector representation is passed through a multi-layer perceptron to produce the final classification logits. The entire framework is trained end-to-end using a supervised learning paradigm. The objective is to minimize the cross-entropy loss between the predicted error category and the ground truth label provided in the training dataset. To prevent the model from overfitting to one dominant modality, we introduce a regularization term that penalizes large discrepancies in the gradient updates between the log and dump feature extraction branches. The traffic diagnosis architecture described by Lin [18] motivates an ablation that separates temporal encoding from the final fault classifier. For log and memory-dump triage, this would mean withholding either event chronology or cross-source links while retaining the same root-cause labels. The overall loss function is defined as: Another relevant line of investigation concerns deep incomplete multi-view network semi-supervised multi-label learning with unbiased loss [19].

$$L = -\sum_{i=1}^{C} y_i log(haty_i) + \lambda|\theta|^2$$

Here, C represents the total number of distinct error categories, y_i is the binary indicator of the correct class, $haty_i$ is the predicted probability, θ represents the trainable parameters of the network, and λ controls the strength of the L2 regularization.

6. Experimental Setup

6.1 Dataset Compilation

To evaluate the efficacy of the proposed cross-source triage framework, we compiled a comprehensive dataset of software faults from a large-scale, open-source distributed cloud infrastructure project. Dapper provides distributed request tracing, a complementary evidence source when a failure spans several services rather than a single process image [20]. The dataset consists of 50,000 distinct crash incidents collected over a period of two years. For each incident, we extracted the core memory dump generated at the time of failure and the preceding 10,000 lines of application logs from the node where the crash occurred. The ground truth labels were derived from the issue tracking system, where human engineers had manually investigated the crashes, identified the root causes, and assigned them to specific bug identifiers. We categorized these bugs into 150 unique error classes, ranging from simple null pointer dereferences and array out-of-bounds errors to complex race conditions and distributed state inconsistencies.

The dataset was partitioned chronologically to simulate a realistic operational deployment scenario. The first 18 months of data, comprising approximately 40,000 incidents, were designated as the training set. The subsequent two months, containing 5,000 incidents, were utilized for validation and hyperparameter tuning. The final two months of data served as the hold-out testing set to evaluate the generalization capabilities of the trained models. For a broader comparison, related research addresses chinese-english bilingual question answering system using cross-lingual pre-trained models [21].

6.2 Baselines and Metrics

We compared our proposed multi-modal framework against several state-of-the-art unimodal baselines to quantify the value of cross-source integration. The baselines included a Log-Only Transformer model, which classified errors based purely on the sequential log embeddings, and a Dump-Only Graph Network, which relied exclusively on the structural analysis of the faulting stack traces. Furthermore, we implemented a Naive Fusion baseline, which simply concatenated the output vectors of the Log-Only and Dump-Only models without the sophisticated co-attention alignment mechanism.

Table 2Categorization of error signatures utilized in the experimental evaluation

Error Category	**Predominant Artifact Dependency**	**Complexity Level**
Memory Corruption	High reliance on Dump Signatures	Low
Network Timeout	High reliance on Log Sequences	Medium
Race Condition	Requires Cross-Source Alignment	High
Config Mismatch	Requires Cross-Source Alignment	High

The performance of the models was evaluated using standard multi-class classification metrics: Precision, Recall, and the F1-score, calculated using macro-averaging across all 150 error classes to ensure that rare, complex bugs were weighted equally with common, simple crashes. We also measured the Mean Reciprocal Rank to assess the model's ability to rank the correct root cause within its top predictions, a critical metric for triage engineers who typically review the top three to five suggestions provided by automated tools. For a broader comparison, related research addresses geometry-enhanced attentive multi-view stereo for challenging matching scenarios [22].

7. Results and Discussion

7.1 Quantitative Performance Evaluation

The empirical results demonstrate a substantial performance advantage for the proposed cross-source framework over all unimodal and naive fusion baselines. On the hold-out testing set, our multi-modal architecture achieved a macro-averaged F1-score of 0.89, significantly outperforming the Log-Only model at 0.72 and the Dump-Only model at 0.76. The Naive Fusion model achieved an F1-score of 0.81, indicating that while simply combining data sources provides a benefit, the advanced co-attention alignment mechanism is crucial for fully exploiting the synergistic potential of the multi-modal data. Sentence-BERT provides a sentence-embedding approach for comparing the semantic content of textual records [23]. Sun and colleagues examine hyperbolic embedding for joint user and community alignment [24]. The superiority of the cross-source approach was most pronounced in the Mean Reciprocal Rank metric. Our framework achieved a score of 0.94, meaning that the correct root cause was almost always listed as the first or second prediction. In contrast, the Dump-Only model struggled significantly with ambiguous crash signatures. For instance, in scenarios where multiple different logical errors ultimately cascaded into the same final out-of-memory exception, the Dump-Only model could not distinguish between the root causes, resulting in a low Mean Reciprocal Rank of 0.68 for that specific subset of errors. Policy retrieval supplies context to the claims-screening explanations of Huang et al. [1]. For system incidents, evidence retrieval should instead expose the relevant trace fragment and configuration, enabling reviewers to distinguish a useful ranked lead from a confirmed defect.

Table 3Comprehensive performance comparison across different diagnostic methodologies

Methodology	Precision	Recall	F1-Score
Log-Only Transformer	0.74	0.71	0.72
Dump-Only Graph Net	0.79	0.74	0.76
Naive Concatenation	0.82	0.80	0.81
Cross-Source Fusion	0.91	0.88	0.89

7.2 Deep Dive into Cross-Source Synergy

To understand the mechanics driving this performance increase, we conducted an ablation study and a qualitative analysis of the attention weights generated by the fusion module. The analysis revealed that the framework effectively utilizes logs to disambiguate identical memory dumps. In one prominent case study from the test set, two separate incidents exhibited the exact same stack trace, crashing deep within a generic database connection library. The Dump-Only model confidently, but incorrectly, grouped them into the same issue bucket. The literature also considers fast bellman updates for wasserstein distributionally robust MDPs [25].

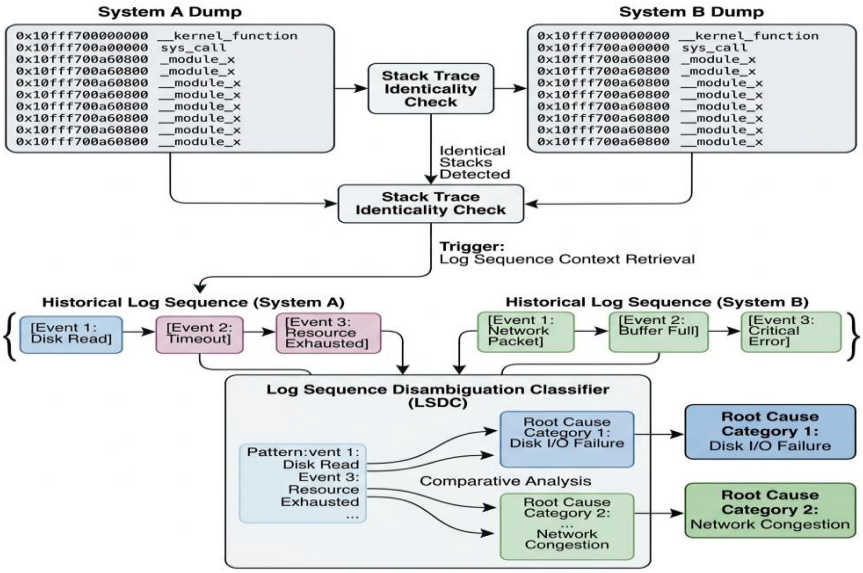


Figure 2Detailed diagnostic flow diagram illustrating the disambiguation process where two identical memory dump stack traces are correctly classified into different root cause categories by leveraging divergent temporal patterns detected in their respective his

However, the cross-source model correctly identified them as distinct bugs. By analyzing the co-attention weights, we observed that for the first incident, the model heavily attended to log messages generated three seconds prior to the crash indicating a specific invalid user payload being processed by an upstream microservice. For the second incident, the model attended to logs indicating a sudden spike in thread creation, pointing to a resource exhaustion attack. The unified diagnostic framework synthesized the contextual "why" from the logs with the deterministic "where" from the dump, achieving an accurate triage decision that neither source could provide independently. The literature also considers resolving task objective conflicts in unified model via Task-Aware Mixture-of-Experts [26].

7.3 Limitations and Practical Considerations

Despite the highly promising results, several limitations must be addressed before widespread industrial adoption. The primary constraint is computational overhead. Processing large memory dumps and executing Transformer-based inference over extensive log sequences requires significant computational resources, which may introduce latency into real-time triage pipelines. Optimization techniques, such as knowledge distillation or model quantization, will be necessary to deploy this framework in edge computing environments or highly constrained cloud instances. Related work has addressed brain-inspired computing accelerated by memristor technology [27].

Furthermore, the framework's reliance on supervised learning implies that it requires a substantial corpus of historically labeled data to achieve optimal performance. For newly deployed applications or systems undergoing rapid architectural changes, accumulating sufficient training data may be challenging. While the model demonstrates robust generalization capabilities, zero-day vulnerabilities or entirely novel failure modes may still result in misclassification, necessitating fallback mechanisms that alert human operators when the model's confidence scores fall below a defined threshold. Wang and colleagues examine decorate cross-lingual pre-training model as Non-Autoregressive neural machine translation [28]. Uncertainty-guided cross-head co-training for semantic segmentation [29] provides a methodological comparator for learning from limited labels, with software-specific validation needed before accepting pseudo-labeled incidents.

8. Conclusion

8.1 Summary of Contributions

The increasing scale and complexity of distributed software architectures have rendered manual error triage obsolete, demanding sophisticated automated diagnostic solutions. This paper introduced a novel cross-source error triage framework designed to synergistically process unstructured application logs and structured memory dump signatures. By developing specialized feature extraction pipelines for each modality and employing an advanced co-attention mechanism for semantic alignment, we established a robust mathematical foundation for multi-modal software diagnostics. The proposed architecture addresses the inherent limitations of isolated, single-source analysis, bridging the contextual gap between historical execution sequences and terminal crash states. Huang and colleagues examine modality asymmetries and circuit dynamics in modern transformers [30]. Through extensive empirical evaluation on a realistic dataset of software faults, we definitively demonstrated the superiority of the cross-source methodology. The unified framework not only achieved higher precision and recall across a diverse spectrum of error categories but also proved highly effective at disambiguating complex, state-dependent bugs that confound traditional analysis tools. By automatically correlating the "how" and "why" provided by logs with the "where" provided by memory dumps, the framework significantly reduces the cognitive burden on reliability engineers and accelerates the mean time to resolution for critical system failures.

8.2 Avenues for Future Research

The success of this initial multi-modal architecture opens several promising avenues for future research in software reliability engineering. One primary direction is the integration of additional diagnostic modalities. Incorporating time-series telemetry data, such as CPU utilization, memory pressure, and network throughput metrics, could provide a third dimension of environmental context, further enhancing the model's ability to diagnose resource-related faults. Related work has addressed adaptive weakly supervised Cross-Modal reliability learning for resource-constrained proxy-state recognition [31]. Counterfactual auditing of multi-signal LiDAR segmentation in CARTN [32] motivates testing whether a diagnostic decision changes when one telemetry channel is removed or corrupted. Another critical area for exploration is the transition from supervised classification to self-supervised or unsupervised anomaly detection. Developing techniques to pre-train the cross-source alignment models on vast quantities of unlabeled operational data could mitigate the dependency on historically labeled datasets, allowing the framework to adapt more rapidly to novel software deployments and emerging failure patterns. Finally, researching mechanisms for automated patch generation or self-healing systems based on the precise root cause localization provided by this triage framework represents the ultimate goal of autonomous software operations. Building secure, verifiable bridges between automated diagnostics and automated remediation will be essential for realizing the next generation of highly resilient computing infrastructures. Feature attribution helps interpret the disclosure anomalies [9]. For cross-source triage, a useful extension would identify whether an alert is driven by log frequency, dump structure or their disagreement, allowing engineers to inspect the relevant source directly. Hyperdimensional uncertainty fusion in autonomous perception [33] suggests a useful extension here: retain source-specific confidence when combining log and dump evidence, rather than treating every diagnostic signal as equally reliable.

References

Huang, D., Song, L., Chen, Y., He, S., Wang, R., Shen, H., & Chu, W. (2026). AnExplainable AI Engineering Framework for Claims-Only First-Stage Provider Audit Triage Using SHAP-Guided Hybrid Retrieval-Augmented Generation. Machine Learning and Knowledge Extraction, 8(9), 279.

Fan, J., Rosu, P., Wang, A. T., Carin, L., & Cheng, X. (2026). In context semi-supervised learning. In Proceedings of the International Conference on Learning Representations. https://openreview.net/forum?id=lqrpmqrTnH

Lyu, S., Mao, S., & Ma, Y. (2025). Unified Long Video Inpainting and Outpainting via Overlapping High-Order Co-Denoising. arXiv preprint arXiv:2511.03272.

Yao, Z., Hawi, P., Aitharaju, V., Mahishi, J., & Ghanem, R. (2026). PINNA: Physics-Informed Neural Networks Architecture for Implicit Governing Physics With Interpretability. International Journal for Numerical Methods in Engineering, 127(17), e70399.

Sang, Y. (2026, March). Adaptive Quantization Strategies for Robust ML Inference Under Distribution Shift. In Proceedings of the 2026 5th International Conference on Cyber Security, Artificial Intelligence and Digital Economy (pp. 362-368).

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google file system. In Proceedings of SOSP.

Xiaoteng Ma, Yiqin Yang, Hao Hu, Qihan Liu, Jun Yang, Chongjie Zhang, Qianchuan Zhao, and Bin Liang. Offline reinforcement learning with value-based episodic memory. arXiv preprint
arXiv:2110.09796, 2021b.

Guo, F., Xu, K., Zhang, Z., Zhou, H., Chen, G., Hu, J., ... & Mo, H. (2025). Battery SOH prediction under different conditions via MBLSTM and iTransformer with anomaly detection and explainability. IEEE Open Journal of the Computer Society.

Sun, F., He, S., Wang, R., Ke, L., Shen, H., & Liao, Q. (2026). Modeling Structural Deviation in 10-K Risk Factors: A Semantic Anomaly Detection and Explainable AI Approach. Risks, 14(4), 87.

Li, Y. (2026, June). AI-augmented automated triage for system error reporting using log and memory dump analysis. In 2026 7th International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE) (pp. 572-577). IEEE.

Qiu, M., Wu, L., & Wang, S. (2026). CyberDQFormer: A Knowledge Graph-Enhanced Transformer Framework for Security-Aware Data Quality Anomaly Detection and Autonomous Repair Against Network Attacks. Fundamental Scientific Reports in Multidisciplinary Areas, 2(3), 34-52.

Guo, F., Tao, S., Zhang, J., Zhou, H., Ji, J. J., Dong, D., & Mo, H. (2026). Frequency Spectral Prior Guided Graph Learning for Degradation Decoupling in Reliable Battery SOH Estimation. Green Energy and Intelligent Transportation, 100446.

Wang, J., Xu, Y., Li, Y., Khargonekar, P., & Faruque, M. A. A. (2026). CRUISE: Vision-Language Model-Guided Uncertainty-Aware Cross-Modal Sensor Fusion for Robust Autonomous Driving. arXiv preprint arXiv:2608.09202.

Zhang, J., Luo, H., Yang, H., Xu, X., Wu, Q., Shi, Y., ... & Wang, J. (2023). Neuraldome: A neural modeling pipeline on multi-view human-object interactions. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 8834-8845).

Li, Q., Luo, T., Jiang, M., Jiang, Z., Hou, C., & Li, F. (2025, April). Semi-supervised multi-view multi-label learning with view-specific transformer and enhanced pseudo-label. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 39, No. 17, pp. 18430-18438).

Lu, L., Su, J., Wang, J., & Deng, X. (2026). AMSPINN: anisotropic and multiscale cross-attention physics-inspired neural networks for turbulence prediction. Engineering Applications of Computational Fluid Mechanics, 20(1), 2628951.

Wu, J., Chen, S., Heo, I., Gutfraind, S., Liu, S., Li, C., ... & Sharps, M. (2025). Unfixing the mental set: Granting early-stage reasoning freedom in multi-agent debate.

Lin, Y. (2024). Design of urban road fault detection system based on artificial neural network and deep learning. Frontiers in neuroscience, 18, 1369832.

Li, Q., Luo, T., Jiang, M., Liao, J., & Jiang, Z. (2024, October). Deep incomplete multi-view network semi-supervised multi-label learning with unbiased loss. In Proceedings of the 32nd ACM International Conference on Multimedia (pp. 9048-9056).

Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., et al. (2010). Dapper, a large-scale distributed systems tracing infrastructure. Technical Report, Google.

Mo Z. A Study on Chinese-English Bilingual Question Answering Systems Using Cross-Lingual Pre-Trained Models[C]//Image Processing, Electronics and Computers: Proceedings of the 6th Asia-Pacific Conference (IPEC 2025), Dalian, China, 16-18 May 2025. 1 Oliver's Yard, 55 City Road, London, EC1Y 1SP: SAGE Publications, 2025: 1715-1724.

Liu, Y., Cai, Q., Wang, C., Yang, J., Fan, H., Dong, J., & Chen, S. (2024). Geometry-enhanced attentive multi-view stereo for challenging matching scenarios. IEEE Transactions on Circuits and Systems for Video Technology, 34(8), 7401-7416.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Proceedings of EMNLP-IJCNLP.

Sun, L., Zhang, Z., Zhang, J., Wang, F., Du, Y., Su, S., & Yu, P. S. (2020, November). Perfect: A hyperbolic embedding for joint user and community alignment. In 2020 IEEE International Conference on Data Mining (ICDM) (pp. 501-510). IEEE.

Yu, Z., Dai, L., Xu, S., Gao, S., & Ho, C. P. (2023). Fast bellman updates for wasserstein distributionally robust mdps. Advances in Neural Information Processing Systems, 36, 30554-30578.

Zhang, J., & Tang, H. (2025). Resolving Task Objective Conflicts in Unified Model via Task-Aware Mixture-of-Experts. arXiv preprint arXiv:2506.03591.

Liu, Chenchen, Fuqiang Liu, and Hai Li. "Brain-inspired computing accelerated by memristor technology." In Proceedings of the 4th ACM International Conference on Nanoscale Computing and Communication, pp. 1-6. 2017.

Wang, Y., He, S., Chen, G., Chen, Y., & Jiang, D. (2022, December). XLM-D: Decorate cross-lingual pre-training model as non-autoregressive neural machine translation. In Proceedings of the 2022 conference on empirical methods in natural language processing (pp. 6934-6946).

Fan, J., Gao, B., Jin, H., & Jiang, L. (2022). UCC: Uncertainty guided cross-head co-training for semi-supervised semantic segmentation. In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 9937–9946). IEEE. https://doi.org/10.1109/CVPR52688.2022.00971

Huang, Y., Roth, K., Bouniot, Q., Xu, W., & Akata, Z. (2026). Dissecting Multimodal In-Context Learning: Modality Asymmetries and Circuit Dynamics in modern Transformers. arXiv preprint arXiv:2601.20796.

Meng, X., & Lin, J. (2026). AWCRL: Adaptive Weakly Supervised Cross-Modal Reliability Learning for resource-constrained proxy-state recognition. Pattern Recognition, 114998.

Meng, X., & Lin, J. (2026). CARTN: Counterfactual Audit and Repair for Trust-Aware Multi-Signal Streaming LiDAR Segmentation. Information Fusion, 104809.

Chen, L., Wang, J., Mortlock, T., Khargonekar, P., & Al Faruque, M. A. (2025, June). Hyperdimensional uncertainty quantification for multimodal uncertainty fusion in autonomous vehicles perception. In 2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 22306-22316). IEEE.