K-LAD: Knowledge-Enhanced Log Anomaly Detection via Syntax-Aware Verification

Lizhu Mi[1,2], Hongbin Zhang[2*], Lu Li[2], Wei Gu[2], Feng Tian[2], Chen Ling[2*]
[1]School of Artificial Intelligence and Computer Science, North China University of Technology, Beijing 100144, China
[2]Center for Information Research, Academy of Military Sciences, Beijing 100142, China
[*]Corresponding authors: 13671190606@163.com (Hongbin Zhang), Bitlingchen@163.com (Chen Ling)

***Abstract*—Large Language Models (LLMs) in anomaly detection often exhibit a Defensive Over-reaction on imbalanced data, while standard retrieval mechanisms relying on generic semantic models (e.g., BERT) suffer from a Semantic-Syntactic Gap by treating logs purely as natural language. To address these challenges, we propose K-LAD, a data-efficient framework grounded in the insight that logs are weakly-structured code derivatives. K-LAD employs a coarse-to-fine cascade architecture: (1) A High-Recall Anomaly Candidate Generator (LoRA-tuned LLaMA-3 with Focal Loss) designed to maximize anomaly capture; and (2) A Syntax-Aware Knowledge Verification stage utilizing UniXcoder to rigorously correct false positives via machine syntax alignment. Extensive experiments on HDFS, BGL, and Thunderbird demonstrate that K-LAD achieves SOTA performance using only 2,000 training samples, empirically validating the superiority of modeling logs via code-derived syntax over traditional NLP approaches.**

***Index Terms*—Log anomaly detection, Large language models, Fine-tuning, AIOps, RAG**

I. Introduction

The exponential growth in scale and complexity of modern IT environments has made the transition towards Artificial Intelligence for IT Operations (AIOps) [1] an imperative for ensuring service reliability [2]. Within this ecosystem, anomaly detection stands as a pivotal component, serving as the critical first line of defense for system stability. As discrete records of system runtime states, system logs contain crucial information regarding failure patterns and execution logic [3]. Consequently, automated log anomaly detection has emerged as a research hotspot in both academia and industry.

Early log analysis methods primarily relied on statistical features (e.g., PCA, IM), before evolving into deep learning-based sequence modeling. Pioneering works such as DeepLog [4] and LogAnomaly [5] modeled logs as natural language sequences via LSTMs, while LogRobust [6] and LogBERT [7] introduced attention mechanisms to capture long-range dependencies. Despite their progress, these supervised methods face a critical bottleneck: a heavy reliance on log parsing tools. Consequently, parsing errors propagate downstream, and when facing evolving log patterns (Concept Drift), these models often fail due to limited semantic generalization capabilities [2].

Recently, breakthroughs in Large Language Models (LLMs) for understanding unstructured text have brought a paradigm shift to log analysis [8], [9]. Trained on massive corpora, LLMs possess powerful contextual reasoning capabilities, theoretically enabling them to bypass the dependency on explicit log parsing. However, applying general-purpose LLMs directly to industrial-grade log anomaly detection faces two severe challenges:

- **False Positive Bias and Hallucination:** This is a core challenge. In real-world operational scenarios where anomaly samples are extremely scarce, fine-tuning often causes LLMs to exhibit a defensive over-reaction. They tend to misclassify benign warnings or rare but normal system behaviors as failures due to hallucination. This phenomenon results in high recall but extremely low precision, generating massive false alarms and increasing operator fatigue [2], [10].
- **Domain Knowledge Gap:** General-purpose LLMs lack runtime contextual knowledge of specific systems. Without external references, it is difficult for the model to distinguish between routine updates and potential risks.

To mitigate the aforementioned issues, Retrieval-Augmented Generation (RAG) [11] has emerged as a promising solution, aiming to constrain LLM outputs by retrieving similar historical logs as context [11], [12]. However, standard retrieval mechanisms often employ the Dense Passage Retrieval (DPR) paradigm [13] with general-purpose semantic models (e.g., BERT). In our preliminary exploratory experiments, we identified that generic DPR models tend to overemphasize synonyms or literal similarity, whereas in the log domain, distinguishing events relies heavily on machine syntax rather than pure literal meaning.

We posit that log templates are not merely unstructured text but weakly-structured derivatives of source code statements (e.g., `printf` or `logger.info`), functioning as intermediate representations between natural language and program execution logic. They are akin to function-level comments describing program operations via implicit parameters. Standard NLP models trained on general text struggle to distinguish these subtle structures. For instance, two logs might be literally similar (e.g., both containing `Connection failed`), yet one represents a normal retry mechanism while the other signals a fatal deadlock, distinguished only by minute differences in parameter structure. Our experiments indicate that general NLP models cannot effectively capture these code-syntax-like differences, leading to the retrieval of irrelevant

context. In contrast, UniXcoder [14], pre-trained on aligned code-text corpora, is uniquely suited to model this machine semantics, generating highly discriminative embeddings for precise retrieval.

In this paper, we propose K-LAD (Knowledge-enhanced Log Anomaly Detection), a data-efficient framework operating on a generate-then-verify paradigm. K-LAD integrates a High-Recall Anomaly Candidate Generator (LoRA-tuned LLaMA-3 with Focal Loss) to capture all potential anomalies, with a Syntax-Aware Knowledge Verification mechanism (UniXcoder-based retrieval) to rigorously filter false positives. By modeling logs via code-derived syntax rather than pure natural language, we utilize structural knowledge to correct LLM hallucinations bi-directionally.

Our main contributions are summarized as follows:

- We propose a coarse-to-fine framework that harmonizes the reasoning power of LLMs with the evidentiary support of retrieval, effectively resolving the high false-positive dilemma in few-shot log analysis.
- We empirically validate the hypothesis that code pre-trained models (e.g., UniXcoder [14]) outperform NLP models for retrieval-based log verification, demonstrating that capturing syntactic structures is key to accurate log retrieval.
- Extensive experiments on HDFS, BGL, and Thunderbird datasets confirm that K-LAD achieves SOTA performance using only 2,000 training samples, outperforming baselines.

II. Related work

This section traces the evolution of log anomaly detection through three technological paradigms, analyzing the specific limitations of existing methods to position the novelty of our proposed K-LAD framework.

A. Discriminative Sequence Modeling and Parsing Reliance

Early research primarily approached log analysis as a sequence classification problem. DeepLog [4] pioneered the use of Long Short-Term Memory (LSTM) networks to model log sequences similarly to natural language sentences, detecting anomalies by predicting the probability of the next log template. Building on this, LogAnomaly [5] introduced template vectors and count vectors to simultaneously capture semantic and frequency deviations. To address the instability of log data, LogRobust [6] proposed an attention-based Bi-LSTM model to enhance robustness against noise. Despite outperforming traditional statistical methods (e.g., PCA), these discriminative approaches face a critical bottleneck: a heavy dependency on upstream log parsers (e.g., Drain). Parsing errors inevitably propagate to the downstream detection model, creating a single point of failure. Furthermore, RNN-based architectures struggle with parallelization and often fail to capture long-range complex semantic dependencies effectively.

B. Pre-trained Language Models and Semantic Limitations

With the advent of Transformer architectures, researchers began leveraging Pre-trained Language Models (PLMs) to enhance log semantic representation. LogBERT [7] adapts the Masked Language Modeling (MLM) task for self-supervised learning, enabling the capture of bidirectional contextual relations. NeuralLog [15] further attempts to eliminate the parsing dependency by processing raw log messages directly via Transformers. Additionally, PLELog [16] addresses label scarcity through a semi-supervised method based on probabilistic label estimation. While these methods improve semantic understanding, they suffer from a fundamental domain bias: they treat system logs as ordinary natural language. However, logs are inherently machine-generated semi-structured text containing numerous machine-specific identifiers (e.g., memory addresses, error codes) and rigid execution logic. Directly applying NLP-based pre-trained models often neglects the underlying code syntax, leading to insufficient discrimination in complex failure scenarios where distinct events share similar literal semantics.

C. Generative Reasoning and Retrieval-Enhanced Strategies

Recently, Large Language Models (LLMs) have been introduced to AIOps due to their strong few-shot reasoning capabilities. LogPrompt [17] explores prompt engineering to guide LLMs for zero-shot detection, while LogLLaMA [18] employs reinforcement learning to align LLMs with normal log patterns. However, applying general-purpose LLMs to industrial-grade log detection faces the challenge of hallucination. Due to the extreme scarcity of anomaly samples, LLMs tend to exhibit a defensive over-reaction, misclassifying unseen normal behaviors as failures. To mitigate this, retrieval-based strategies have been proposed to constrain LLM outputs using historical context. For instance, RAPID [19] utilizes retrievers to identify relevant historical logs as reference. Distinct from these works, K-LAD addresses the Semantic-Syntactic Gap prevalent in standard retrieval mechanisms. Unlike existing methods that rely on generic semantic models (e.g., BERT) for retrieval, we utilize a code pre-trained model (UniXcoder) to construct a syntax-aware verifier. By harmonizing generative reasoning with code-derived structural evidence, K-LAD effectively solves the hallucination problem while maintaining high data efficiency.

III. Methodology

This section details the proposed K-LAD framework, a data-efficient, coarse-to-fine anomaly detection architecture designed to harmonize the generative reasoning capabilities of LLMs with the deterministic evidence of syntax-aware retrieval. As illustrated in Figure 1, the workflow comprises four integral modules: Log Preprocessing, Historical Knowledge Repository Construction, and the two core detection stages: High-Recall Anomaly Candidate Generation (Stage 1) and Syntax-Aware Knowledge Verification (Stage 2). The following subsections describe each component in detail.

A. Overview of K-LAD Framework

To address the Defensive Over-reaction of Large Language Models (LLMs) on imbalanced log data and the Semantic-Syntactic Gap in retrieval mechanisms, K-LAD adopts a Cascade Architecture with Retrieval-Augmented Verification. The process operates in two core sequential stages:

1) **High-Recall Anomaly Candidate Generation (Stage 1):** This stage utilizes a LoRA-tuned LLaMA-3-8B model as a High-Recall Anomaly Candidate Generator. By incorporating Focal Loss [20], it functions as a High-Sensitivity Scanner, prioritizing Recall to ensure that every potential anomaly is captured as a candidate, deliberately tolerating a higher false positive rate to avoid missed detections.
2) **Syntax-Aware Knowledge Verification (Stage 2):** This stage employs UniXcoder as a Syntax-Aware Verifier. Operating on Historical Knowledge Repositories, it performs an Always-on deterministic verification for every candidate generated in Stage 1. Through a Relative Confidence Competition mechanism, it leverages structural evidence to bi-directionally correct both False Positives (hallucinations) and False Negatives, ensuring the final decision is grounded in factual syntax patterns.

B. Log Preprocessing

Raw system logs are semi-structured text streams containing both fixed templates and variable runtime parameters. To transform them into model-readable inputs, we execute a two-step preprocessing pipeline.

First, we utilize a log parser (e.g., Drain [21]) to map raw log messages into structured log templates. This process extracts invariant keywords (representing static code logic) while masking variable runtime parameters (such as IP addresses and timestamps) with generic tokens (e.g., <*>). Let L denote a raw log message; the parsing function P maps it to a unique template $e = P(L)$, which serves as a normalized event signature.

Subsequently, since single log messages often lack sufficient semantic context, we group these templates into sequences to capture temporal execution dependencies. For session-based datasets like HDFS, logs are aggregated by specific identifiers (e.g., `Block_ID`), where a sequence is represented as $x = [e_1, e_2, \ldots, e_k]$. Conversely, for window-based datasets like BGL, we apply a fixed or sliding window strategy, where a sequence x consists of logs occurring within a specific time or count interval. The constructed sequence x then serves as the input for the subsequent anomaly detection stages.

C. Stage 1: High-Recall Anomaly Candidate Generation

The first stage serves as a "High-Recall Anomaly Candidate Generator." We employ LLaMA-3-8B and adapt it to the log domain using Low-Rank Adaptation (LoRA) [22].

1) Semantic Instruction Tuning: To transition the model from general text generation to domain-specific log analysis, we structure the training data in an instruction-following format (I, x, y). Here, I represents the guiding instruction (e.g., Analyze the execution logic of the following log sequence...), x denotes the input log sequence, and y is the target label. The optimization objective is to maximize the conditional probability of the label token:

$$\mathcal{L}_{gen} = -\sum_{t=1}^{T} \log P(y_t | x, I; \Theta_{LoRA}) \tag{1}$$

2) Addressing Imbalance with Focal Loss: In production environments, anomalies are extremely scarce, leading to severe class imbalance. Standard Cross-Entropy loss is often dominated by the majority class (normal logs), causing the model to exhibit a false negative bias (i.e., tending to predict everything as normal). To counteract this, we incorporate Focal Loss into the LLM fine-tuning process:

$$\mathcal{L}_{FL}(p_t) = -\alpha(1 - p_t)^{\gamma} \log(p_t) \tag{2}$$

where p_t represents the model's estimated probability for the ground-truth class, α is a weighting factor to balance class importance, and γ is the focusing parameter. Specifically, γ down-weights easy negative samples, forcing the model to concentrate on sparse, hard-to-classify anomalies. This strategy ensures that Stage 1 prioritizes Recall, outputting a set of Anomaly Candidates (denoted as $\hat{y}_{LLM}$) for subsequent verification.

D. Historical Knowledge Repository Construction

To empower the framework with reasoning capabilities beyond model parameters, K-LAD implements a rigorous Knowledge Engineering process. We transform unstructured historical logs into computable, retrievable Expert Knowledge Repositories through three core dimensions:

1) Knowledge Abstraction: Raw logs contain significant runtime noise (e.g., IPs, timestamps). To extract Invariant Syntactic Patterns, we apply log parsing to decouple logs into invariant Templates and variable Parameters. This step achieves noise reduction, abstracting raw operational data into stable event signatures that reflect the static logic of the source code.

2) Syntax-Aware Representation: To capture the deep logical correlations between logs, we map discrete log templates into a continuous high-dimensional vector space. Unlike standard NLP models that focus on natural language semantics (often leading to the Semantic-Syntactic Gap), we utilize the code-specific model UniXcoder [14] to extract features:

$$v_x = \text{UniXcoder}(\text{Template}(x)) \in \mathbb{R}^d \tag{3}$$

where d is the dimension of the embedding vector (e.g., 768). Since UniXcoder is pre-trained on ASTs and aligned code-text corpora, the generated vector v_x encodes the underlying Code Execution Logic. This effectively transforms human-readable templates into machine-computable Syntax-Aware Knowledge, bridging the gap between textual description and program behavior.

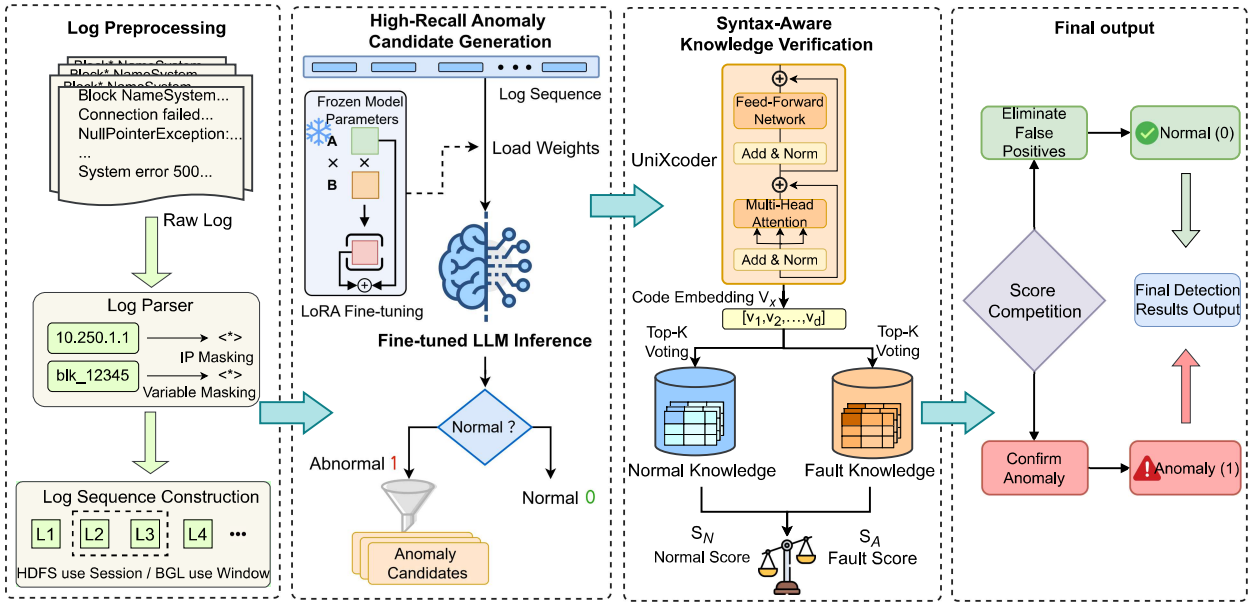

Fig. 1. **Overview of the K-LAD Framework.** The architecture operates in a coarse-to-fine manner: (1) **Stage 1 (High-Recall Anomaly Candidate Generation)** employs a LoRA-tuned LLaMA-3 to scan for potential anomalies with high sensitivity; (2) **Stage 2 (Syntax-Aware Knowledge Verification)** utilizes UniXcoder to retrieve structural evidence from historical repositories, performing deterministic bi-directional correction to filter out false positives.

3) Structured Knowledge Organization: We organize the sedimented semantic knowledge into two decoupled repositories to support efficient retrieval:

- **Normal Knowledge Repository ($\mathcal{K}_N$):** Stores semantic vectors of historical normal operations, representing the system's baseline knowledge.
- **Fault Knowledge Repository ($\mathcal{K}_A$):** Stores semantic vectors of the few-shot known anomalies, representing the system's expert experience.

E. Stage 2: Syntax-Aware Knowledge Verification

To mitigate LLM hallucinations (false positives) and rescue missed anomalies (false negatives), this stage functions as a deterministic judge. It is composed of two distinct components working in synergy:

- **Syntax-Aware Verifier:** Utilizes UniXcoder to capture the rigorous syntactic structure of log sequences, providing a machine-readable representation.
- **Historical Knowledge Repositories:** Leverage the pre-constructed Normal ($\mathcal{K}_N$) and Fault ($\mathcal{K}_A$) Repositories as ground-truth evidence.

1) Bi-directional Top-K Knowledge Retrieval: For an input sample x, we perform a Top-K retrieval to find the most structurally similar candidates from both repositories. Let $\mathcal{N}_K(v_x, \mathcal{K})$ denote the set of K nearest neighbors of the query vector v_x in repository $\mathcal{K}$. We define the retrieval scores S_N and S_A as the maximum Cosine Similarity within these Top-K neighbors:

$$S_N = \max_{k \in \mathcal{N}_K(v_x, \mathcal{K}_N)} \text{CosSim}(v_x, k) \tag{4}$$

$$S_A = \max_{k \in \mathcal{N}_K(v_x, \mathcal{K}_A)} \text{CosSim}(v_x, k) \tag{5}$$

where $\text{CosSim}(\cdot)$ denotes the cosine similarity function defined as $\frac{A \cdot B}{\|A\|\|B\|}$. This Top-$K$ strategy ($K = 5$ in our experiments) ensures that the verification is robust to local noise in the vector space.

2) Relative Confidence Competition Mechanism: We adopt a Relative Competition Mechanism to arbitrate the final decision. The system compares the retrieval scores against the confidence thresholds τ_N and τ_A:

$$y_{final} = \begin{cases} 0, & \text{if } (S_N > \tau_N) \wedge (S_N > S_A) \\ 1, & \text{if } (S_A > \tau_A) \wedge (S_A > S_N) \\ \hat{y}_{LLM}, & \text{otherwise} \end{cases} \tag{6}$$

where τ_N and τ_A are the strict thresholds for the normal and anomaly repositories.

This logic ensures robust bi-directional correction:

- **Deterministic Correction (Normal):** If the sample strictly matches normal patterns (Condition 1), we correct the label to 0, effectively filtering false positives (hallucinations).
- **Deterministic Correction (Anomaly):** If the sample strictly matches known fault patterns (Condition 2), we correct the label to 1, rescuing potential false negatives.
- **Fallback to Generator:** In ambiguous cases where retrieval evidence is insufficient to override the generator, we trust the LLM's prediction $\hat{y}_{LLM}$.

IV. Experiments

In this section, we conduct extensive experiments on three widely used public log datasets to evaluate the effectiveness

TABLE I
STATISTICS OF EXPERIMENTAL DATASETS

Dataset	Total Logs	Anomalies	Ratio (%)	Train Size
HDFS	11,175,629	327,446	2.93%	2,000
BGL	4,747,963	348,460	7.34%	2,000
Thunderbird	10,000,000	353,794	3.54%	2,000

of K-LAD. We compare our proposed framework with seven state-of-the-art (SOTA) baseline methods and provide a detailed analysis of the contribution of each module.

A. Datasets and Preprocessing

To demonstrate the strong generalization capabilities of our model in low-resource scenarios, we adopt an extremely data-efficient setting. Specifically, for all datasets, we sample only 2,000 log sequences for training and use the remainder for testing, verifying the robustness of K-LAD under few-shot conditions.We selected three benchmark datasets from different sources:

- **HDFS [23] Dataset:** Generated by running benchmark workloads on a Hadoop Distributed File System cluster. The dataset consists of 11,175,629 raw log messages. Partitioned by unique `Block_ID`, it yields a total of 575,061 sequences, which are labeled as normal or anomalous based on expert rules. The dataset exhibits a natural imbalance with an anomaly ratio of approximately 2.93%.
- **BGL [24] Dataset:** Collected from the Blue Gene/L supercomputer system at Lawrence Livermore National Laboratory (LLNL), containing 4,747,963 log messages. Unlike HDFS, we employ a fixed window strategy to construct sequences. This dataset is notably noisy, containing 348,460 log entries identified as anomalies based on explicit alert categories.
- **Thunderbird [24] Dataset:** Collected from the Thunderbird supercomputer system at Sandia National Laboratories (SNL). To ensure experimental efficiency, we utilize a subset comprising the first 10,000,000 consecutive logs. Consistent with BGL, we apply a fixed window mechanism to construct log sequences. This subset presents a complex detection challenge with an anomaly ratio of 3.54%.

The statistics of these datasets are summarized in Table I.

B. Implementation Details

All experiments were conducted on a server equipped with two NVIDIA Tesla V100 GPUs (32GB), running on Ubuntu 20.04 LTS with CUDA 12.0.

- **Model Architecture:** We adopt the LLaMA-3-8B architecture as the backbone and employ LoRA (Low-Rank Adaptation) to achieve parameter-efficient fine-tuning (PEFT) [25], significantly reducing memory overhead while retaining generalization capabilities.
- **Training Parameters:** During fine-tuning, the learning rate is initialized at 2e-4. We utilize the AdamW optimizer [26] coupled with a Cosine Annealing scheduler to dynamically adjust the learning rate, facilitating stable convergence.

C. Evaluation Metrics

To comprehensively and objectively measure the performance of anomaly detection models, we follow standard conventions in the field and employ Precision, Recall, and F1-score as core metrics. In binary classification, we define:

- **TP (True Positive)**: Anomaly sequences correctly predicted as anomalies.
- **FP (False Positive)**: Normal sequences incorrectly predicted as anomalies (False Alarm).
- **FN (False Negative)**: Anomaly sequences incorrectly predicted as normal (Missed Detection).

The metrics are calculated as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{8}$$

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{9}$$

Precision reflects the reliability of alarms; high precision implies fewer false alarms, thereby mitigating "alert fatigue" for operators. Recall measures the system's coverage, ensuring that critical failures are not overlooked. The F1-score, as the harmonic mean of Precision and Recall, serves as the primary metric for evaluating performance in our highly imbalanced experimental scenarios.

D. Baseline Comparison

To verify the effectiveness and superiority of K-LAD in few-shot, low-resource scenarios, we conducted comprehensive comparative experiments against seven widely adopted log anomaly detection baselines. These methods span three distinct paradigms: DeepLog [4], LogAnomaly [5], LogRobust [6], LogBERT [7], NeuralLog [15], PLELog [16], and the recent LLM-based method LogPrompt [17].

To ensure a fair comparison, we reproduced all baseline methods using their official open-source implementations or standard community benchmarks, strictly adhering to their recommended hyperparameter settings. We categorize these baselines into three groups based on their underlying architectures:

1) **RNN/LSTM-based Methods (DeepLog, LogAnomaly, LogRobust):** These approaches rely on Recurrent Neural Networks (RNNs) or LSTMs to model log sequences and detect anomalies based on violation of sequential patterns.
2) **Transformer/Pre-training Methods (LogBERT, NeuralLog, PLELog):** These methods leverage Transformer [27] architectures or Masked Language Modeling (MLM) to capture long-range dependencies and semantic relationships. While they mitigate the dependency on labeled data to some extent, they inherently treat logs

TABLE II
PERFORMANCE COMPARISON WITH BASELINE METHODS ON HDFS, BGL, AND THUNDERBIRD DATASETS

Method	HDFS			BGL			Thunderbird		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
DeepLog	0.768	0.705	0.735	0.828	0.897	0.861	0.873	0.896	0.884
LogAnomaly	0.868	0.718	0.785	0.761	0.731	0.741	0.877	0.956	0.914
LogRobust	0.953	0.898	0.924	0.847	0.753	0.797	0.558	0.871	0.680
LogBERT	0.782	0.679	0.727	0.854	0.905	0.879	0.976	0.965	0.970
PLELog	0.905	0.936	0.920	0.826	0.817	0.821	0.810	0.781	0.795
NeuralLog	**0.974**	0.968	0.971	0.917	0.905	0.910	0.803	0.928	0.861
LogPrompt	0.213	0.834	0.339	0.259	0.813	0.393	0.256	0.985	0.406
K-LAD (Ours)	0.967	**0.979**	**0.973**	**0.994**	**0.970**	**0.982**	**0.981**	**0.992**	**0.986**

TABLE III
ABLATION STUDY RESULTS ON HDFS, BGL, AND THUNDERBIRD DATASETS

Setting	HDFS			BGL			Thunderbird		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
w/o Stage 2 (LLM Only)	0.610	**0.980**	0.752	0.718	0.997	0.835	0.755	**0.996**	0.859
w/o Stage 1 (Verifier Only)	0.913	0.623	0.741	0.968	0.845	0.902	0.955	0.862	0.906
K-LAD (w/ BERT)	0.113	0.976	0.202	0.158	**1.000**	0.274	0.174	0.954	0.295
K-LAD (Full)	**0.967**	0.979	**0.973**	**0.994**	0.970	**0.982**	**0.981**	0.992	**0.986**

as pure natural language, often overlooking the rigorous syntactic structures derived from source code.

3) **LLM-based Methods (LogPrompt):** A recent approach utilizing the zero-shot reasoning capabilities of Large Language Models via prompt engineering. Although training-free, our experiments reveal that without external knowledge augmentation, pure LLM reasoning is prone to severe hallucinations, resulting in extremely low precision (< 0.3) on complex datasets like BGL and Thunderbird.

Analysis of Results: As shown in Table II, K-LAD outperforms existing baselines in overall performance.

- **Dominance in Complex Scenarios:** On the semantically complex and noisy BGL (F1=0.982) and Thunderbird (F1=0.986) datasets, K-LAD achieves State-of-the-Art (SOTA) results, leading the runners-up (e.g., NeuralLog and LogBERT). This performance leap is directly attributed to our Stage 2 Syntax-Aware Knowledge Verification mechanism, which precisely distinguishes logs that are semantically similar but logically different.
- **Robustness in Low-Resource Settings:** Compared to traditional deep learning models (e.g., DeepLog, LogRobust) that suffer from overfitting when trained on limited data, K-LAD maintains high stability and precision with only 2,000 training samples. This validates the data efficiency of our "Coarse-to-Fine" paradigm, where the LLM handles generalization while the retrieval module handles memorization.
- **Mitigating Hallucinations:** Compared to the pure LLM-based method LogPrompt, K-LAD improves the F1-score by several folds (from $\sim$0.4 to $>$0.9). This empirically proves that relying solely on probabilistic LLM reasoning is insufficient for AIOps; external Retrieval-Augmented Verification is essential to ground the model's predictions in factual history, thereby meeting the high precision requirements of industrial operations.

E. Ablation Study

To verify the necessity of the two-stage cascade architecture and the crucial role of syntax-aware retrieval in K-LAD, we conducted a detailed ablation study comparing four settings:

1) **w/o Stage 2 (LLM Only):** Removing Stage 2, using only the fine-tuned LLM for prediction.
2) **w/o Stage 1 (Verifier Only):** Removing Stage 1, directly using UniXcoder to retrieve and classify test samples.
3) **K-LAD (w/ BERT):** Replacing the Syntax-Aware Verifier (UniXcoder) in Stage 2 with a standard NLP retriever (bert-base-uncased), while keeping other components identical.
4) **K-LAD (Full)**: The complete two-stage cascade architecture with UniXcoder.

The results are presented in Table III.

- **LLM False Positive Bias**: Using only Stage 1 results in high Recall but significantly drops Precision (e.g., only 0.61 on HDFS). This confirms that LLMs tend to exhibit a defensive over-reaction on imbalanced data, flagging many normal logs as potential anomalies.
- **Impact of Syntax-Awareness (BERT Failure):** As shown in the table, replacing UniXcoder with BERT results in catastrophic performance drops (e.g., F1 0.202

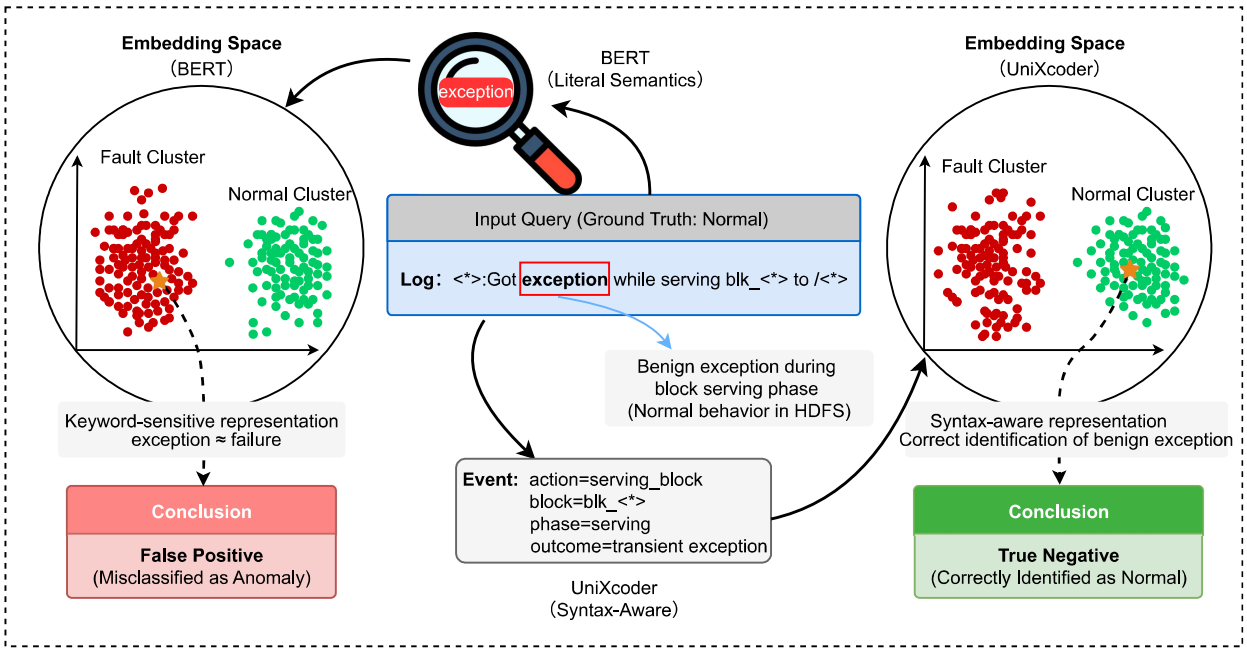

Fig. 2. Case Study: Embedding Space Visualization. Given a benign log with ambiguous keywords (e.g., `exception`), the NLP-based retriever (BERT) is misled by literal semantics, retrieving fault patterns and causing a False Positive. In contrast, the Syntax-Aware Verifier (UniXcoder) captures the underlying execution logic (`serving...to...`), correctly mapping it to the normal pattern and filtering out the false alarm.

on HDFS). Notably, BERT's precision (≈0.11) is even lower than the LLM Only baseline. This indicates that generic NLP models focus on literal semantics (e.g., matching the word "failed"). When processing Stage 1's false positives, BERT retrieves structurally irrelevant fault evidence due to keyword matching, thus reinforcing the false alarms instead of correcting them. This empirically proves the Semantic-Syntactic Gap hypothesis.

- **Complementary Advantage**: The full architecture with UniXcoder combines the high recall of Stage 1 with the high precision of Stage 2. For instance on HDFS, introducing the syntax-aware Stage 2 boosts Precision from 0.61 to 0.967 while maintaining a high Recall of 0.979. This demonstrates that K-LAD successfully filters the noise, leveraging both the reasoning power of LLMs and the deterministic syntactic knowledge of code retrieval.

F. Case Study: Visualizing the Semantic-Syntactic Gap

To intuitively understand why K-LAD outperforms BERT-based methods, we visualized the embedding space of ambiguous logs in Figure 2. The selected log message from the HDFS dataset is: `<*>: Got exception while serving blk_<*> to /<*>`. This log contains the alarming keyword `exception` but typically indicates a transient network fluctuation during a standard block serving process (a benign event).

- **BERT's Failure (Literal Trap):** As illustrated in the left branch of Fig. 2, BERT focuses heavily on the token `exception`. In the general semantic space, this word is strongly correlated with system failures. Consequently, the log's embedding drifts into the Fault Knowledge Repository (Red Cluster), leading the retrieval module to retrieve irrelevant crash reports as evidence. This results in a False Positive, exacerbating the defensive over-reaction.
- **UniXcoder's Success (Syntactic Disambiguation):** Conversely, the right branch shows that UniXcoder perceives the log as a procedural structure: `ServeBlock(State=Exception)`. It recognizes the syntactic pattern of `serving...to...` as a valid data transfer operation. Despite the `exception` keyword, the syntax-aware embedding correctly aligns with the Normal Knowledge Repository (Green Cluster). This allows K-LAD to deterministically identify the log as benign, effectively rescuing the LLM's hallucination.

This qualitative analysis empirically confirms our hypothesis: modeling logs via code-derived syntax is essential for distinguishing logic-variant anomalies from semantically similar noise.

V. Conclusion

In this paper, we presented K-LAD, a novel Knowledge-enhanced Log Anomaly Detection framework that successfully bridges the gap between the probabilistic reasoning capabilities of LLMs and the deterministic evidence provided by syntax-aware retrieval.

We first identify the inherent defensive overreaction of general-purpose LLMs when fine-tuned on imbalanced data, alongside the semantic-syntactic gap that constrains standard retrieval mechanisms when processing semi-structured logs. To overcome these bottlenecks, we propose a data-efficient cascaded architecture comprising a high-recall anomaly candidate generator optimized via focus loss and a syntax-aware knowledge validator based on UniXcoder.

Therefore, integrating syntax-aware structural knowledge is crucial for precise anomaly validation. Experimental results confirm that K-LAD achieves state-of-the-art performance even under data-constrained scenarios, effectively suppressing false positives (hallucination phenomena) while maintaining high recall.

In future work, we plan to explore extending K-LAD to online streaming scenarios and investigating multi-modal operations analysis that integrates system metrics with logs.

REFERENCES

[1] Y. Dang, Q. Lin, and P. Huang, "Aiops: real-world challenges and research innovations," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 4–5.

[2] M. De la Cruz Cabello, T. P. Sales, and M. R. Machado, "Aiops for log anomaly detection in the era of llms: A systematic literature review," *Intelligent Systems with Applications*, p. 200608, 2025.

[3] L. Zhang, T. Jia, M. Jia, Y. Wu, A. Liu, Y. Yang, Z. Wu, X. Hu, P. S. Yu, and Y. Li, "A survey of aiops for failure management in the era of large language models," *arXiv preprint arXiv:2406.11213*, 2024.

[4] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[5] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, vol. 19, no. 7, 2019, pp. 4739–4745.

[6] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 807–817.

[7] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 international joint conference on neural networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[8] W. Guan, J. Cao, S. Qian, J. Gao, and C. Ouyang, "Logllm: Log-based anomaly detection using large language models," *arXiv preprint arXiv:2411.08561*, 2024.

[9] H. Guo, J. Yang, J. Liu, J. Bai, B. Wang, Z. Li, T. Zheng, B. Zhang, J. Peng, and Q. Tian, "Logformer: A pre-train and tuning pipeline for log anomaly detection," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 38, no. 1, 2024, pp. 135–143.

[10] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM computing surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[11] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[12] T. Zhang, Z. Jiang, S. Bai, T. Zhang, L. Lin, Y. Liu, and J. Ren, "Rag4itops: A supervised fine-tunable and comprehensive rag framework for it operations and maintenance," *arXiv preprint arXiv:2410.15805*, 2024.

[13] V. Karpukhin, B. Oguz, S. Min, P. S. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering." in *EMNLP (1)*, 2020, pp. 6769–6781.

[14] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," *arXiv preprint arXiv:2203.03850*, 2022.

[15] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 492–504.

[16] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1448–1460.

[17] Y. Liu, S. Tao, W. Meng, F. Yao, X. Zhao, and H. Yang, "Logprompt: Prompt engineering towards zero-shot and interpretable log analysis," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, 2024, pp. 364–365.

[18] Z. Yang and I. G. Harris, "Logllama: Transformer-based log anomaly detection with llama," *arXiv preprint arXiv:2503.14849*, 2025.

[19] G. No, Y. Lee, H. Kang, and P. Kang, "Training-free retrieval-based log anomaly detection with pre-trained language model considering token-level information," *Engineering Applications of Artificial Intelligence*, vol. 133, p. 108613, 2024.

[20] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[21] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.

[22] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models." *ICLR*, vol. 1, no. 2, p. 3, 2022.

[23] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns of console logs," in *2009 ninth IEEE international conference on data mining*. IEEE, 2009, pp. 588–597.

[24] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE, 2007, pp. 575–584.

[25] Y. Huang, T. Ma, K. Yang, and Z. Zhang, "Finsent-distillq: A distilled large language model with chain-of-thought fine-tuning for financial sentiment analysis," *Journal of Intelligent Information Systems*, pp. 1–37, 2026.

[26] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.