

Seminar Report

Log-Based Anomaly Detection: From Classical Methods to Foundation Models

Mohamed Basuony
MatrNr: 12647187

Supervisor: Sadegh Keshtkar

Georg-August-Universität Göttingen
Institute of Computer Science

September 30, 2025

Abstract

System logs are noisy, high-volume, and evolve as software changes, breaking many anomaly detectors trained under "stable log" assumptions. This survey examines log-based anomaly detection from classical feature methods to modern foundation-model pipelines, unifying results across Hadoop Distributed File System (HDFS), Blue Gene/L (BGL), and Thunderbird datasets. We review classical baselines (PCA, invariants), sequence models (DeepLog, OC4Seq, LogRobust) for behavioral and local/global anomalies, transformer families that capture semantics and structure (LogAnomaly, LogBERT, UniLog, LOGFORMER/PEFT), and Large Language Model (LLM)-based approaches spanning prompting, supervised fine-tuning, and Reinforcement Learning (RL)-aligned policies (LogPrompt, GPT-3FT, LogGPT, LogLLaMA). We complement modeling with parsing and data concerns—Drain and Loghub-2.0 reveal template-level pitfalls affecting downstream detection, while AnomalyGen addresses dataset coverage via LLM-assisted log synthesis. Our synthesis indicates: (i) parameter-efficient adapters deliver strong cross-domain accuracy with low tuning cost; (ii) RL Top-K alignment improves recall/precision over pure language-modeling losses; and (iii) under instability, targeted fine-tuning outperforms prompt-only strategies. We provide a practical taxonomy, harmonized cross-dataset comparison, and deployment guidance balancing accuracy, robustness, and operational cost for evolving production logs.

Declaration on the use of ChatGPT and comparable tools in the context of examinations

In this work I have used ChatGPT or another AI as follows:

- Not at all
- During brainstorming
- When creating the outline
- To write individual passages, altogether to the extent of % of the entire text
- For the development of software source texts
- For optimizing or restructuring software source texts
- For proofreading or optimizing
- Further, namely: -

I hereby declare that I have stated all uses completely.

Missing or incorrect information will be considered as an attempt to cheat.

Contents

List of Tables	iv
List of Figures	iv
List of Abbreviations	v
1 Introduction	1
1.1 Contributions	1
1.2 Report Structure	1
2 Background and Datasets	2
2.1 Log Structure and Processing Challenges	2
2.2 Benchmark Datasets and Evaluation Protocols	2
3 Classical and Sequential Methods	3
3.1 Classical Feature-Based Approaches	3
3.2 Sequence Models and RNN Architectures	3
4 Transformer-Based Detection	4
4.1 Semantic and Contextual Modeling	4
4.2 Parameter-Efficient and Unified Approaches	4
5 Large Language Model Approaches	5
5.1 Prompt-Based Detection with Explanations	5
5.2 Fine-Tuning for Log Evolution	5
5.3 Reinforcement Learning Alignment	6
6 Data Processing and Synthesis	6
6.1 Parsing at Production Scale	6
6.2 Synthetic Data Generation	7
7 Empirical Analysis and Performance Comparison	7
7.1 Cross-Dataset Performance Analysis	7
7.2 Cost-Accuracy Trade-off Analysis	8
7.3 Reproducibility Notes (for Figures 3 and 4)	8
7.4 Protocol Considerations and Validity Threats	9
8 Deployment Guidelines and Practical Recommendations	10
8.1 Method Selection Framework	10
8.2 Method Family vs. Robustness and Ops Characteristics	11
8.3 Production Pipeline Architecture	11
8.4 Operational Considerations	12
9 Conclusion and Future Directions	12
References	14

List of Tables

- 1 Performance comparison of log anomaly detection methods across datasets 8
- 2 Method family vs. robustness and ops characteristics (concise view). 11

List of Figures

- 1 RL Top-K Training Process 4
- 2 AnomalyGen Workflow 7
- 3 Cost vs Accuracy Trade-off 9
- 4 F1 Leaderboard 10

List of Abbreviations

AI	Artificial Intelligence
BGL	Blue Gene/L
CFG	Control Flow Graph
CoT	Chain of Thought
FGA	F1 of Group Accuracy
FT	Fine-Tuning
FTA	F1 of Template Accuracy
GNN	Graph Neural Network
GPT	Generative Pre-trained Transformer
GRU	Gated Recurrent Unit
HDFS	Hadoop Distributed File System
HPC	High-Performance Computing
LLM	Large Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
MLKP	Masked Log-Key Prediction
NLP	Natural Language Processing
OC-SVM	One-Class Support Vector Machine
PCA	Principal Component Analysis
PEFT	Parameter-Efficient Fine-Tuning
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SHAP	SHapley Additive exPlanations
SVDD	Support Vector Data Description
TF-IDF	Term Frequency-Inverse Document Frequency
VHM	Variational Hypersphere Modeling
XAI	Explainable Artificial Intelligence

1 Introduction

Modern software systems emit massive, heterogeneous logs whose formats and semantics evolve with every release. Models trained under “stable log” assumptions often fail when templates change, parameters shift, or pipelines reconfigure, causing false alarms or missed incidents in production. Three critical challenges recur: (i) **unstructuredness** (free-text, variable tokens), (ii) **instability** (new/altered templates and shuffled/duplicated events), and (iii) **data scarcity** for novel behaviors. Any practical detector must combine robust structuring, semantics-aware representation, and adaptation to drift.

The field has evolved through distinct phases. Early systems converted logs into count vectors and applied classical Machine Learning (ML) (PCA, invariants, one-class methods): simple and fast, but brittle to order/semantics and drift [1, 2, 3]. Sequence models like DeepLog learned next-event behavior with parameter/time-series heads, improving recall while remaining sensitive to parsing choices [4]. Robust Recurrent Neural Network (RNN) variants (OC4Seq, LogRobust) addressed local/global anomalies and instability via one-class losses and semantic vectors with attention [3, 5].

Transformer families then captured richer semantics and context through joint sequential+quantitative modeling with template embeddings (LogAnomaly) [6], bidirectional self-supervision plus compactness (LogBERT) [7], unified multi-task frameworks (UniLog) [8], and parameter-efficient tuning with Log-Attention (LOGFORMER) [9]. Most recently, LLM-based detectors span zero/few-shot prompting with natural-language rationales (LogPrompt) [10], targeted fine-tuning for unstable logs (GPT-3FT) [11], and reinforcement-learning alignment with Top-K inclusion rewards (LogGPT, LogLLaMA) [12, 13].

1.1 Contributions

This survey provides three key contributions:

1. A consolidated view of robustness to instability and unseen templates across method families.
2. A harmonized cross-dataset summary of reported F1 scores for widely cited baselines and recent LLM models.
3. Practical trade-off guidance balancing accuracy/robustness against cost/latency using quantitative analysis.

We emphasize evaluation protocol differences and note where results are not directly comparable, enabling readers to adapt methods confidently to evolving production environments. We also *restrict all citations to the provided source list from the course materials (21 sources total)* and introduce no external references.

1.2 Report Structure

The remainder of this report is organized as follows: Section 2 provides background on log structure and benchmark datasets. Sections 3–5 survey classical, transformer, and LLM-based approaches respectively. Section 6 covers data processing and synthesis. Section 7

presents empirical performance comparisons (with protocol notes and reproducibility detail). Section 8 provides deployment guidelines, and Section 9 concludes with future directions.

2 Background and Datasets

2.1 Log Structure and Processing Challenges

System logs combine **stable tokens** (event verbs, system identifiers) with **variable parameters** (IDs, paths, latencies, timestamps). Minor code changes can rewrite templates or reorder events, creating **instability** that breaks detectors trained on previous formats. Effective pipelines must (i) separate **structure** from **parameters**, (ii) learn **semantics** beyond surface identifiers, and (iii) tolerate **drift** and **unseen templates**.

Drain has emerged as the de-facto online parser, using a fixed-depth parse tree that routes messages by length and stable prefixes to create templates in streaming time [14]. While Drain demonstrates strong accuracy and runtime across multiple datasets (HDFS, BGL, High-Performance Computing (HPC), ZooKeeper, Proxifier), large-scale re-evaluation in Loghub-2.0 reveals that classical message-level metrics can overestimate parser quality [15]. Rare and parameter-heavy templates remain challenging for most parsers, and template-level F1 scores (F1 of Group Accuracy (FGA)/F1 of Template Accuracy (FTA)) expose significant gaps where strong grouping on parameter-light logs can collapse when five or more parameters appear.

2.2 Benchmark Datasets and Evaluation Protocols

We focus on three widely-used public corpora for concrete comparisons:

- **HDFS** (distributed file system): ~ 11 M logs with anomalies defined at block/session level, typically grouped by session ID [4, 9, 6].
- **BGL** (supercomputer logs): ~ 4 – 5 M logs with long sequences, usually time-windowed (60s) due to lack of natural session keys [6, 4, 7, 3].
- **Thunderbird** (HPC): tens of millions of entries with very large key spaces, typically windowed, stressing long-context models [7, 3, 15].
- **LOGEVOL-Hadoop** (software evolution): two versions (v2 \rightarrow v3) for studying unstable logs in GPT-3 fine-tuning vs. GPT-4 prompting [11].

Critical evaluation considerations include: (1) **Label granularity**—some works label sequences rather than individual lines, requiring aggregation that can inflate false positives; (2) **Thresholding**—unsupervised scores require validation-tuned thresholds that materially shift F1; (3) **Parsing effects**—missed/merged templates change vocabularies, degrading template-ID-based models; (4) **Instability protocols**—synthetic perturbations reveal that prompt-only LLMs degrade while fine-tuned/RL-aligned models maintain robustness; (5) **Compute costs**—families differ significantly in training and serving requirements.

3 Classical and Sequential Methods

3.1 Classical Feature-Based Approaches

Classical methods convert logs to count/TF-IDF vectors or simple statistics, then apply linear subspace or one-class boundaries (PCA, Invariant Mining, One-Class Support Vector Machine (OC-SVM), Isolation Forest). These approaches offer fast deployment with small footprints, making them suitable as first-line detectors. However, they lose sequential order and semantic information, proving brittle to template changes and unseen events [1, 2, 3].

Performance results show supervised variants can achieve strong scores on curated settings (Decision Trees ≈ 0.998 F1 on HDFS), while unsupervised classical baselines generally lag deep models (PCA F1 ≈ 0.77 on HDFS, OC-SVM ≈ 0.36) [1]. These methods particularly struggle on BGL/Thunderbird where sequences are long and key spaces are large, highlighting the importance of sequence modeling and semantic understanding.

3.2 Sequence Models and RNN Architectures

DeepLog treats template IDs as tokens, training stacked Long Short-Term Memory (LSTM)s to predict the next key and flagging anomalies when observed keys fall outside top- g predictions [4]. The approach includes parameter heads for numeric modeling (times, counts) with validation-based thresholding, workflow exposure for diagnosis, and online updates from analyst feedback. DeepLog demonstrates high F1 on HDFS (≈ 0.96) and OpenStack-I (≈ 0.98), with online updates on BlueGene/L lifting precision from ≈ 0.16 to ≈ 0.82 – 0.88 . However, it remains vulnerable to unseen templates and parsing noise, requiring careful window/session grouping and top- g selection.

OC4Seq addresses multi-scale anomaly detection by learning both global (full sequence) and local (sliding-window) embeddings with Gated Recurrent Unit (GRU)s, training both with Deep-Support Vector Data Description (SVDD)-style hypersphere loss [3]. Normal sequences cluster near centers c (global) and c_L (local), with distance-based scoring indicating anomalies. OC4Seq achieves state-of-the-art RNN performance: HDFS 0.976, RUBiS 0.985, BGL 0.747 F1, outperforming PCA, OC-SVM, Invariant Mining, DeepLog, and Deep SVDD. Nevertheless, dependence on template tokens and hyperparameter sensitivity (window size, local/global balance α) remain limitations.

LogRobust replaces brittle template IDs with semantic event vectors (FastText + Term Frequency-Inverse Document Frequency (TF-IDF) over preprocessed tokens), then applies attention Bi-LSTM over sequences [5]. This semantic approach dampens the impact of logging changes, noisy parsing, and minor text edits while supporting incremental fine-tuning. On synthetic HDFS with new/altered templates (up to 20% replacement), F1 remains ≈ 0.89 – 0.95 ; with sequence perturbations (remove/duplicate/shuffle up to 20%), F1 ≈ 0.95 . Real Microsoft service deployment (weekly updates) achieves F1 ≈ 0.81 , outperforming classical baselines by $\sim 30\%$ absolute.

4 Transformer-Based Detection

4.1 Semantic and Contextual Modeling

Transformer models advance log anomaly detection through semantic template embeddings, bidirectional context, and parameter-efficient adaptation. **LogAnomaly** replaces brittle template IDs with template2Vec semantic embeddings that respect synonym/antonym relations, learning both sequential patterns and quantitative invariants in a unified LSTM framework [6]. For unseen templates, the approach approximates vectors online without retraining. Semantic proximity maintains appropriate distances between related events (e.g., keeping “link up/down” semantically distant despite surface similarity), while the count-invariant channel reduces false positives when frequencies shift under normal behavior. Results show F1 ≈ 0.95 – 0.96 on HDFS/BGL with markedly fewer false alarms versus ID-based models.

LogBERT treats log sequences as text, using Transformer encoders for masked log-key prediction (Masked Log-Key Prediction (MLKP)) and the **Volume of Hypersphere Minimization (VHM)** objective (Deep-SVDD-inspired) on [DIST] token embeddings [7, 2]. Bidirectional attention captures long-range co-occurrence patterns, while the hypersphere term pulls normal sequences together for stable distance-based scoring. Reported performance includes HDFS 0.82, BGL 0.91, Thunderbird 0.97 F1, outperforming RNNs on longer sequences. However, reliance on parsed template tokens and sensitivity to thresholding parameters (g, r) and masking ratios remain considerations.

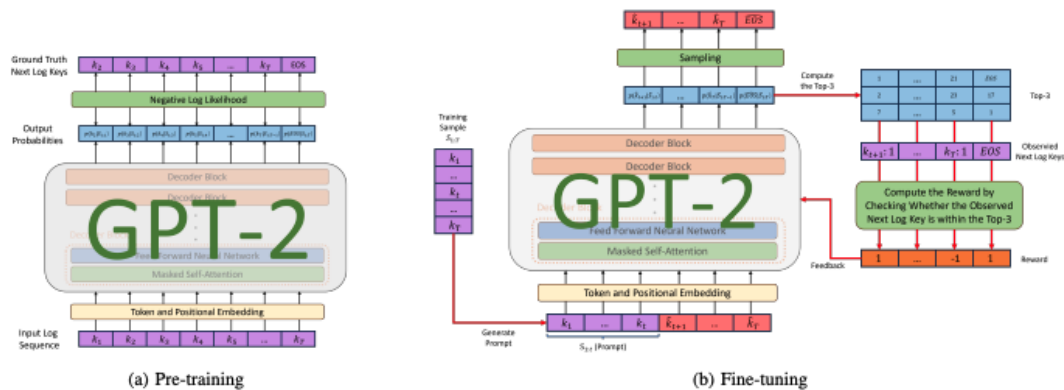


Figure 1: RL Top-K training process showing pre-training and fine-tuning phases with reward mechanism for log anomaly detection.

4.2 Parameter-Efficient and Unified Approaches

LOGFORMER preserves parameter information typically discarded by parsing through Log-Attention (injecting parameters as attention bias) and enables cross-domain adaptation via parallel adapters trained while freezing the main encoder [9]. The two-stage approach pretrains on source logs, then tunes only adapters and classification head on target domains. This achieves cross-domain generalization with $\leq 5\%$ trainable parameters while maintaining state-of-the-art accuracy: HDFS 0.98, BGL 0.97, Thunderbird 0.99 F1. Industrial deployment (GAIA) demonstrates robust performance, with adapters

reducing tuning cost and avoiding catastrophic forgetting while the bias term re-injects parameters into attention rather than ignoring them.

UniLog provides a unified log-Transformer with denoising self-supervision, fine-tuning separate task heads for anomaly detection, failure prediction, summarization, and compression under a single log-to-log interface [8]. Architectural modifications (Pre-LN, LogAct) better accommodate log tokenization and long sequences. Shared pretraining captures both syntax and semantics, enabling transfer learning with reported F1 gains over LogAnomaly/DeepLog for anomaly detection, improved failure prediction, stronger summarization, and enhanced compression rates.

Logsy learns compact normal embeddings while pushing anomalies outward using radial/spherical loss, optionally regularized with auxiliary anomaly data from unrelated systems [16]. Operating without log parsing by tokenizing raw messages, anomaly scores derive simply from embedding norms ($\|z\|$). The objective directly aligns with normal/abnormal separation, with auxiliary negatives providing useful boundary shape. Large-scale HPC log evaluation shows substantial F1 improvements over DeepLog/PCA (Thunderbird ≈ 0.99) with strong precision and few false alarms.

5 Large Language Model Approaches

5.1 Prompt-Based Detection with Explanations

LLM-based detection offers three complementary capabilities: zero/few-shot reasoning with natural-language rationales, targeted fine-tuning for evolving templates, and reinforcement-learning alignment optimizing detection objectives rather than pure language modeling.

LogPrompt applies few-shot prompting with self-prompting, chain-of-thought (Chain of Thought (CoT)), and in-context examples to stabilize outputs, plus format controls for reliable response parsing [10]. Without domain-specific training, LogPrompt achieves parsing F1 ≈ 0.80 across eight datasets and AD F1 = 0.384 (BGL), 0.450 (Spirit), beating trained baselines in online, evolving settings. CoT and few-shot prompt design improve scores by ~ 1.6 – $2\times$ over naive prompts. Complementary work includes ChatGPT for Lustre logs and HuntGPT, which integrate detection dashboards with explainable Artificial Intelligence (AI) (SHapley Additive exPlanations (SHAP)/LIME) for human triage [17, 18].

These prompt-only approaches excel when requiring speed, interpretability, and no training for cold-start domains or incident triage. Limitations include token/context constraints, prompt sensitivity, and lower robustness under heavy instability compared to tuned models.

5.2 Fine-Tuning for Log Evolution

ADUL fine-tunes GPT-3 on labeled stable logs, then applies it to unstable (evolved) logs, comparing against GPT-4 few-shot prompts [11]. On LOGEVOL-Hadoop, fine-tuned GPT-3 achieves F1 = 0.998 (stable) and 0.989 (unstable v2 \rightarrow v3), outperforming CNN/NeuralLog/LogRobust and significantly beating GPT-4 prompt engineering, especially as instability increases (30% sequence changes: 0.938 vs. 0.514 for few-shot GPT-4). By perturbation type, GPT-3Fine-Tuning (FT) handles remove/duplicate operations well,

though shuffle perturbations remain challenging where sequence-order modeling dominates.

This targeted adaptation approach provides optimal robustness-cost balance when logs evolve frequently, requiring more compute than prompting but substantially less than RL-aligned multi-epoch training.

5.3 Reinforcement Learning Alignment

RL-aligned approaches shift objectives from next-token cross-entropy to Top-K inclusion rewards: given context $S_{1:t}$, models are rewarded when true next log keys fall within Top-K predictions, otherwise penalized. **LogGPT** uses Proximal Policy Optimization (PPO) while **LogLLaMA** applies REINFORCE on LLaMA-2 backbone [12, 13]. Figure 1 illustrates the RL training process, showing both pre-training and fine-tuning phases with the Top-K reward mechanism.

Performance results demonstrate consistent improvements: LogGPT (GPT-2 backbone) achieves HDFS 0.901, BGL 0.958, Thunderbird 0.986 F1, with RL adding consistent boosts over pure LM fine-tuning and improving recall through alignment with detection rules. LogLLaMA (LLaMA-2 + RL) reaches HDFS 0.894, BGL 0.959, Thunderbird 0.974 F1, topping transformer/RNN baselines, with RL improving F1 by $\sim 1.5\text{--}5\%$ versus non-RL variants.

The approach addresses a fundamental misalignment: sequence models trained with cross-entropy penalize all rank errors equally, while production detectors care about membership in Top-K operational thresholds. RL directly optimizes this decision boundary, reducing false alarms when multiple plausible next keys exist. However, higher training and serving costs, along with sensitivity to K selection and thresholding, require careful operational consideration.

6 Data Processing and Synthesis

6.1 Parsing at Production Scale

Large-scale evaluations demonstrate that parser quality directly determines detector performance. The Loghub-2.0 study across 14 datasets with millions of lines each shows common message-level scores overestimate performance, while template-level metrics (F1 of Group Accuracy—FGA, Template Accuracy—FTA) expose major failures on rare and parameter-heavy templates [15]. Drain maintains scalability and reliability as an online parser with consistent completion where other parsers time out, though grouping quality degrades as parameter counts increase.

For detectors training on template IDs (DeepLog-style), parser errors and template churn directly propagate to detection performance. Semantic encoders (LogAnomaly, LogRobust, LOGFORMER) show less brittleness to text drift while still benefiting from stable grouping. Practical guidance includes preferring Drain for online systems while instrumenting parser drift monitors (new-template rates, FGA on held-out templates) and reporting template-level metrics alongside F1, especially for parameter-heavy services.

6.2 Synthetic Data Generation

Public corpora often lack event coverage and execution context, limiting generalization [1, 15]. **AnomalyGen** addresses this limitation by combining static program analysis (logging-related call-graph pruning, log-oriented Control Flow Graph (CFG)s) with LLM chain-of-thought to synthesize realistic, annotated log sequences without system execution [19]. Figure 2 demonstrates the four-phase AnomalyGen workflow: generating pruned callgraphs, obtaining enhanced control flow graphs, recursive merging with chain-of-thought verification, and final anomaly log annotation. The approach achieves $\sim 97.5\%$ event coverage, expands events by $38\text{--}95\times$ over benchmarks, and improves downstream F1 by up to $+3.7\%$ when used for augmentation.

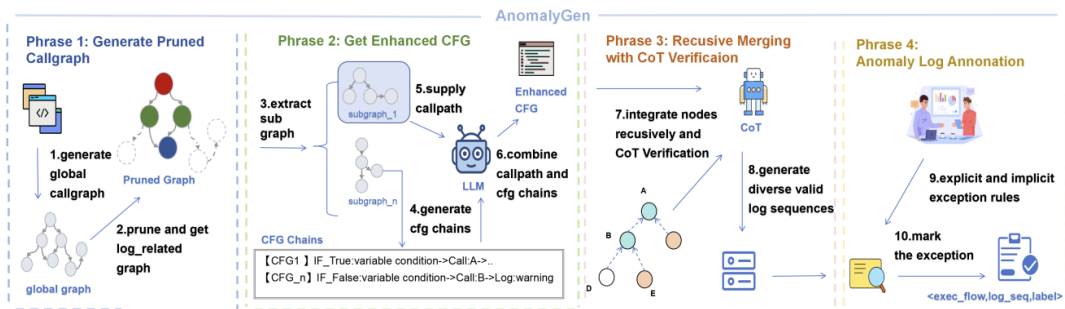


Figure 2: AnomalyGen four-phase workflow for synthetic log generation using static analysis and LLM chain-of-thought.

This synthetic generation proves valuable for domains where collecting labeled incidents is slow or risky. Best practices include using synthesis to balance rare templates and create negative/edge cases for threshold calibration while maintaining human review to vet rules and reduce LLM branch errors, documenting provenance (real vs. synthetic), and evaluating detectors on held-out real logs to avoid overfitting to generator artifacts.

7 Empirical Analysis and Performance Comparison

7.1 Cross-Dataset Performance Analysis

We harmonized reported F1 scores across HDFS, BGL, and Thunderbird datasets, treating HDFS as session-based and BGL/Thunderbird as window-based (typically 60s), following common practice across major studies *within the 21 cited sources* [6, 4, 9, 7, 3, 12, 13]. Results derive from original papers, with baseline numbers from DeepLog/LogAnomaly/LogBERT/OC4Seq source publications, modern LLM methods from unified LogGPT comparisons supplemented by LogLLaMA where available, and LOGFORMER results from its paper (noting protocol differences with LogGPT).

Table 1 presents a comprehensive comparison of key methods across the three primary datasets:

Protocol notes (applies to Table 1 and Figures 3, 4):

- **Granularity:** F1 is computed at *sequence* level (a session or time-window is the unit).

Table 1: Performance comparison of log anomaly detection methods across datasets

Method	HDFS F1	BGL F1	Thunderbird F1	Cost Index	Training Type
DeepLog	0.824	0.861	0.926	3	RNN Sequential
LogAnomaly	0.524	0.867	0.931	4	Semantic LSTM
LogBERT	0.745	0.905	0.963	5	Transformer
OC4Seq	0.808	0.391	0.845	4	One-Class RNN
LOGFORMER	0.980	0.970	0.990	4	PEFT Transformer
LogGPT	0.901	0.958	0.986	9	RL-Aligned LLM
LogLLaMA	0.894	0.959	0.974	9	RL-Aligned LLM

- **Grouping rules:** **HDFS**—session by block ID. **BGL**—sliding time window (5 min in some BERT-based work; 60 s in many LLM works). **Thunderbird**—sliding time window (1 min common).
- **Parsing:** Results follow each paper’s setting: early RNN work often used *Spell*; most Transformer/LLM results use *Drain*. We keep the original parser per source to avoid re-estimation.
- **Decision rules:** **DeepLog/LogGPT/LogLLaMA**—next-event *Top-K/Top-g* inclusion; **LogAnomaly**—joint sequential + count invariants; **LogBERT**—masked prediction with *Top-g* check; **OC4Seq**—one-class distance (global+local hyperspheres); **LOGFORMER**—supervised classifier (pretrain on source, adapter tuning on target).
- **Training data:** Unsupervised/one-class models use *normal-only* for training; RL-aligned models report using small normal subsets (e.g., $\text{few} \times 10^3$ sequences) for policy updates.
- **Harmonization:** We *do not* re-run or re-threshold; values are transcribed from the **21** cited sources. Cross-paper deviations (window size, *K/g*, thresholds) are acknowledged here; hence comparisons are a *reference*, not a single-protocol benchmark.

7.2 Cost-Accuracy Trade-off Analysis

We mapped each method family to a cost/latency index (1–10) combining trainable parameter ratios, context/inference footprint, and RL overhead, plotting against mean F1 over reported datasets (Figure 3). Classical ML (PCA/Invariants) provides the lowest cost with lowest mean F1, suitable as first-line filters. Prompting (LogPrompt-style) requires no training with moderate F1 in online/unstable settings while adding interpretability.

Parameter-Efficient Fine-Tuning (PEFT)/Adapters (LOGFORMER) represent the “knee of the curve” with near-SOTA F1 using $\leq 5\%$ trainable parameters and strong cross-domain transfer. Full fine-tuning (GPT-3FT/ADUL) provides higher cost but best robustness under instability, outperforming prompt engineering as changes increase. RL-aligned LLMs (LogGPT/LogLLaMA) achieve top F1/recall with highest cost/latency, with RL adding consistent 1.5–5% F1 over non-RL variants in equivalent backbones.

The cross-dataset F1 leaderboard (Figure 4) visualizes these results, clearly showing the dominance of transformer and LLM approaches across all three datasets.

7.3 Reproducibility Notes (for Figures 3 and 4)

Mean-F1 aggregation. For each family, mean F1 is the arithmetic mean of the reported F1 values across the datasets covered by that family in the cited papers. We do not re-run or re-threshold any model; all values are transcribed from the **21** sources cited in this report.

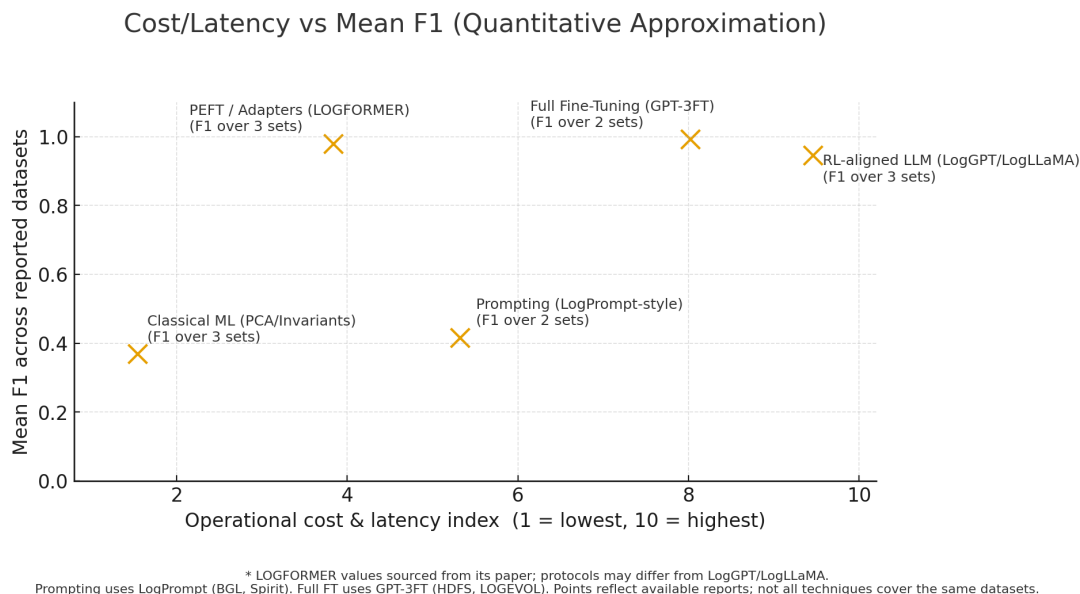


Figure 3: Cost/latency vs mean F1 quantitative approximation showing the trade-off between operational cost and detection accuracy across different method families.

Operational cost & latency index. The x-axis index (1–10) is relative and unitless, derived from: (i) fraction of trainable parameters during adaptation (PEFT/adaptor < full FT < RL-aligned), (ii) expected sequence length / inference context cost (approx. FLOPs), and (iii) presence of policy updates/rollouts vs. prompt-only use. *Concretely, we compute*

$$\text{CostIndex} = w_p \cdot \text{ParamFrac} + w_c \cdot \text{NormContextCost} + w_r \cdot \text{RLOverhead},$$

and linearly rescale to [1, 10] for visualization (weights chosen to reflect practical engineering priority on serving cost).

Grouping/decision policies. HDFS uses session (block) grouping; BGL and Thunderbird use time windows as reported in each source (typically 5 min in BERT-based work, 60 s in many LLM works, 1 min in Thunderbird). Decision policies follow the original papers (Top- K/g , distance thresholds, or classifiers).

7.4 Protocol Considerations and Validity Threats

Protocol mismatches. Some LOGFORMER results use different splits/windowing from LogGPT’s unified evaluation; window sizes (60 s vs. 5 min), Top- K /Top- g , and thresholding can materially shift F1.

Threshold dependence. Reported F1 depends on validation-tuned thresholds; the same model can vary by several points with different K or percentile cutoffs.

Parsing dependencies. Methods assuming template sequences (e.g., DeepLog-style) are sensitive to Drain vs. Spell vs. parser drift; vocabulary changes propagate directly to detection performance.

Coverage imbalance. Not all papers report all datasets; mean-F1 per family thus compares overlapping but not identical subsets. Interpret trends, not absolute ranks.

Limitations and failure modes. Residual risks include (i) *parser drift* that injects spurious vocabulary shifts after software releases; (ii) *synthetic augmentation artifacts*

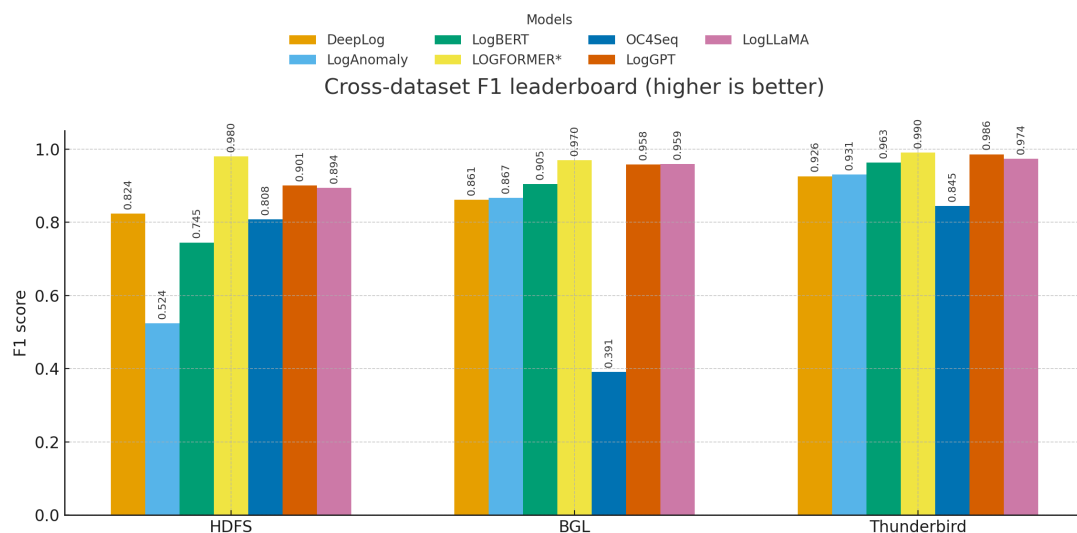


Figure 4: Cross-dataset F1 leaderboard comparing performance of different methods across HDFS, BGL, and Thunderbird datasets.

(e.g., biased token distributions or over-regular control-flow) that can inflate scores if not validated on held-out real incidents; (iii) *shuffle-sensitivity* in fine-tuned LM settings (ADUL) where order perturbations degrade F1 relative to remove/duplicate changes; and (iv) *Top-K sensitivity* in RL-aligned models—too small K reduces recall, too large K increases false negatives at sequence aggregation. Mitigations appear in Section 8.

8 Deployment Guidelines and Practical Recommendations

8.1 Method Selection Framework

Zero labels / need explanations immediately: Few-shot prompting (LogPrompt-style) with strict output formats optimizes for triage and cold-start domains, providing high interpretability despite modest F1 [10, 17, 18].

Tight budget with broad service reuse: PEFT/adapters (LOGFORMER-style) fine-tune $\leq 5\%$ of parameters while maintaining light serving requirements and near-SOTA accuracy with parameter semantics via Log-Attention [9].

Frequent upgrades / unstable logs: Fine-tune compact LLMs on recent stable logs (GPT-3FT/ADUL), which outperform prompt engineering as instability rises, particularly beyond 10–20% perturbation levels [11].

Maximum accuracy/recall on long, diverse sequences: RL-aligned LLMs (Top-K reward; LogGPT/LogLLaMA) provide best overall F1/recall despite higher training and serving costs when tuned appropriately [12, 13].

Security / cross-entity reasoning: Graph methods (Log2vec GNN) or semantic embeddings with graph context address multi-entity attack patterns [20, 21].

Parse-fragile or expensive pipelines: Semantic encoders that mitigate parser errors (LogRobust) or no-parse token models (Logsy) reduce parsing dependencies [6, 5].

Table 2: Method family vs. robustness and ops characteristics (concise view).

Method	Parser dep.	Instability robustness	Labels	Ops cost
DeepLog	High (IDs)	Low–Med (template drift hurts)	Unsupervised (normal-only)	Medium
LogAnomaly	Medium	Med–High (semantic t2v)	Unsupervised	Medium
LogBERT	Medium	Medium (context helps)	Unsupervised	Medium
OC4Seq	Low	Medium (global+local)	Unsupervised (one-class)	Low–Med
LOGFORMER	Low	High (PEFT + Log-Attn)	Supervised (source labels)	Low (adapter FT)
LogGPT	Medium	High (Top- K RL)	Unsupervised + RL reward	High
LogLLaMA	Medium	High (Top- K RL)	Unsupervised + RL reward	High

Notes: Parser dep. = reliance on stable template IDs. Instability robustness = tolerance to new/changed templates and sequence drift. Ops cost = relative training/inference cost (compute + latency).

8.2 Method Family vs. Robustness and Ops Characteristics

The columns of Table 2 summarize the main operational trade-offs observed across families. **Parser dep.** reflects how much a method relies on stable template IDs from a log parser (high values are more brittle to template churn). DeepLog exemplifies high dependency because it models next-event over template IDs [4]; semantic encoders (e.g., LogAnomaly) soften this via template embeddings [6]; PEFT-style transformers (LOGFORMER) further reduce reliance by re-injecting parameters through Log-Attention [9]. **Instability robustness** rates tolerance to evolving templates and sequence perturbations: multiscale one-class learning (OC4Seq) and semantic embeddings (LogAnomaly/LogBERT) increase resilience over plain ID models [3, 7, 6]; PEFT with adapters (LOGFORMER) sustains high robustness across domains [9]; RL-aligned LLMs (LogGPT/LogLLaMA) maintain strong recall under drift by optimizing Top- K inclusion rather than pure cross-entropy [12, 13]. **Labels** indicates whether explicit anomaly labels are needed: most classical/transformer baselines learn from normal-only data (unsupervised or one-class), LOGFORMER typically uses supervised source labels before adapter tuning, and RL-aligned LLMs learn via a task-aligned reward signal without anomaly labels. **Ops cost** coarsely aggregates adaptation footprint and serving latency: adapter-based tuning is light-weight (low), RNN/Transformer baselines are medium, and policy optimization with long contexts is higher. Practically, this means: (i) choose *LOGFORMER* when you need near-SOTA accuracy with tight compute budgets and frequent domain switches; (ii) choose *RL-aligned LLMs* when maximum recall is critical and you can afford higher training/serving cost; (iii) prefer *semantic/one-class* models (LogAnomaly, LogBERT, OC4Seq) when you want robustness over ID-only baselines without full LLM overhead. These qualitative ratings are consistent with the quantitative trends reported in Sections 7 and 7.3.

8.3 Production Pipeline Architecture

1. **Ingestion and Parsing:** Deploy Drain for online, fixed-depth tree parsing while tracking parser drift through new-template rates and FGA/FTA validation metrics [15, 14].
2. **Representation:** Choose between (a) template IDs with semantics (LogAnomaly template2Vec, LogRobust FastText+TF-IDF), (b) raw token processing (Logsy), or

- (c) template + parameter embeddings with Log-Attention (LOGFORMER) [6, 9, 5, 16].
3. **Detection:** Select method family per constraint analysis (RNN/one-class, transformer/PEFT, FT-LLM, RL-LLM) [4, 9, 3, 12, 11, 13].
 4. **Decision Making:** Aggregate per-event scores to sequence labels (session/window) matching dataset protocols; expose Top-K lists or anomaly indices for operators [4, 7, 12].
 5. **Online Learning:** Implement feedback loops (DeepLog online updates), periodic adapter re-tuning (PEFT), or lightweight few-shot prompts for novel behaviors until retraining [4, 9, 10].

8.4 Operational Considerations

Threshold and Parameter Calibration: Select Top-K as vocabulary fraction (30–50%) and validate on held-out normal sequences to meet target false positive rates, with RL-aligned models explicitly optimizing this criterion [4, 12, 13]. For distance/score thresholds (one-class/hypersphere/spherical loss), fit on validation scores using Gaussian tail or quantile methods at desired FPR and re-calibrate after parser or software changes [7, 2, 3, 16].

Instability Hardening: Implement semantic resilience through semantic vectors (LogAnomaly, LogRobust) or parameter re-injection via attention bias (LOGFORMER) [6, 9, 5]. Establish adaptation schedules with adapter-only refreshes (hours) and FT/RL refreshes (less frequent) for services with regular releases. Apply data augmentation (AnomalyGen) to fill rare paths and generate annotated negatives while maintaining human review and real incident validation [19].

Cost and Latency Management: Prefer PEFT for optimal accuracy-cost balance by keeping encoders frozen and scaling adapters per service [9]. Prune context to effective window lengths, cache KV for decoder inference where supported, apply batching and mixed precision (BF16/FP16) with numeric stability guardrails, and implement tiered routing with cheap first-line filters escalating to LLMs only for suspicious windows [1, 9, 10].

9 Conclusion and Future Directions

Log-based anomaly detection has evolved from brittle template-ID matching to sophisticated semantic modeling with foundation model capabilities. The progression from classical methods through RNNs to transformers and LLMs demonstrates clear improvements in handling log instability, semantic understanding, and cross-domain generalization. Parameter-efficient approaches like LOGFORMER represent practical sweet spots for most operational contexts, while RL-aligned LLMs establish new performance frontiers for high-stakes applications requiring maximum recall.

Key findings from our analysis indicate that semantics-aware representations consistently outperform purely syntactic approaches, parameter-efficient tuning provides optimal cost-accuracy trade-offs for cross-domain deployment, and targeted fine-tuning surpasses prompt-only strategies under significant log evolution. The critical importance

of parsing quality, template-level evaluation metrics, and synthetic data augmentation emerges as foundational for reliable production deployment.

Future research directions include: (i) **multimodal anomaly detection** integrating logs with metrics, traces, and topology for comprehensive system monitoring; (ii) **efficient long-context processing** through advanced caching and routing mechanisms maintaining real-time requirements; (iii) **verifiable explainability** combining textual rationales with rigorous attribution methods; and (iv) **standardized benchmarking** protocols encompassing parsing consistency, instability scenarios, and comprehensive cost metrics alongside accuracy measures.

For practitioners, the path forward involves strategic method selection based on operational constraints, robust pipeline architectures incorporating parser monitoring and adaptive thresholding, and careful attention to deployment considerations balancing accuracy, robustness, and cost in evolving production environments. As software systems continue increasing in complexity and scale, log-based anomaly detection must evolve to provide reliable, interpretable, and operationally sustainable monitoring capabilities.

References

- [1] Shilin He et al. “Loghub: A Large Collection of System Log Datasets for AI-based Log Analytics”. In: *arXiv preprint arXiv:2008.06448* (2020).
- [2] Lukas Ruff et al. “Deep SVDD: Deep One-Class Classification”. In: *International Conference on Machine Learning*. 2018, pp. 4393–4402.
- [3] Julien Audibert et al. “OC4Seq: Multi-Scale One-Class RNNs for Discrete Event Sequence Anomaly Detection”. In: *arXiv preprint arXiv:2008.13361* (2020).
- [4] Min Du et al. “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] Xu Zhang et al. “LogRobust: 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.
- [6] Weibin Meng et al. “LogAnomaly: Unsupervised Log-Based Anomaly Detection via Semantic Templates and Joint Modeling”. In: *IJCAI*. 2019, pp. 4735–4742.
- [7] Haixuan Guo, Shuhan Yuan, and Xintao Wu. “LogBERT: Log Anomaly Detection via BERT with Masked Log-Key Prediction and Hypersphere Compactness”. In: *arXiv preprint arXiv:2103.04475* (2021).
- [8] Liang Zhao et al. “UniLog: A Unified Transformer Framework for Log Anomaly Detection, Failure Prediction, Summarization, and Compression”. In: *arXiv preprint arXiv:2112.03159* (2021).
- [9] Shaohan Liu et al. “LOGFORMER: Transformer with Log-Attention and Adapter-based Tuning for Cross-Domain Log Anomaly Detection”. In: *arXiv preprint arXiv:2401.04749* (2024).
- [10] Jinyang Zhao et al. “LogPrompt: Prompt-based, Interpretable and Adaptive Online Log Analysis with LLMs”. In: *arXiv preprint arXiv:2308.07610* (2023).
- [11] Yichen Wang et al. “ADUL: Fine-Tuning GPT-3 for Anomaly Detection on Unstable Logs (vs. Prompted GPT-4)”. In: *arXiv preprint arXiv:2406.07467* (2024).
- [12] Weibin Liu et al. “LogGPT: GPT-based Log Anomaly Detection with RL Top-K Fine-Tuning”. In: *arXiv preprint arXiv:2309.14482* (2023).
- [13] Yu Zhang et al. “LogLLaMA: LLaMA-2 with Top-K REINFORCE for Log Anomaly Detection”. In: *arXiv preprint arXiv:2503.14849* (2025).
- [14] Pinjia He et al. “Drain: An Online Log Parser for Real-Time, Structured Log Extraction”. In: *2017 IEEE International Conference on Web Services (ICWS)*. 2017, pp. 33–40.
- [15] Zhihan Li et al. “Loghub-2.0: A Large-Scale, Template-Level Benchmarking of Log Parsers”. In: *arXiv preprint arXiv:2308.10828* (2023).
- [16] Sasho Nedelkoski et al. “Logsy: Transformer-based Log Anomaly Classification with Spherical Loss and Auxiliary Anomalies”. In: *arXiv preprint arXiv:2008.09340* (2020).

- [17] Liang Chen, Xiaofeng Wang, and Kai Zhang. “*HuntGPT: An XAI + LLM Intrusion Detection Dashboard (Random Forest + SHAP/LIME + GPT)*”. In: *arXiv preprint arXiv:2309.16021* (2023).
- [18] Saurabh Kumar et al. “*ChatGPT-based Anomaly Detection for Parallel File System Logs (Lustre)*”. In: *ACM Digital Library* (2023). DOI: 10.1145/3588195.3595943.
- [19] Jian Liu et al. “*AnomalyGen: LLM-Enhanced Static Analysis for Realistic, Annotated Log Synthesis*”. In: *arXiv preprint arXiv:2504.12250* (2025).
- [20] Fucheng Liu et al. “*Log2vec: Heterogeneous Graph Embeddings for Cyber Threat Detection from Logs*”. In: *arXiv preprint arXiv:1910.10683* (2019).
- [21] Haoyu Li et al. “*Log2Vec: Semantic-Aware Log Representation with OOV Handling*”. In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 2020, pp. 1–9.