JEECS (Journal of Electrical Engineering and Computer Sciences)
Vol. 11, No. 1, June 2026, pp. 23-35
e-ISSN: 2579-5392 p-ISSN: 2528-0260

Self-Supervised Log Anomaly Detection with LogBERT-Style Transformers: Full Empirical Evaluation on a Reproducible SynHDFS Benchmark

Qi Xin

Management Information Systems, University of Pittsburgh, 4200 Fifth Ave, Pittsburgh, PA, USA

Article Info

Article history:
Received: 8 February 2026
Revised: 24 March 2026
Accepted: 9 April 2026

Keyword:
AIOps
BERT
Log anomaly detection
Self-supervised learning
Transformer

Abstract

Log-based anomaly detection is a core problem in AIOps because system logs provide fine-grained evidence of failures, performance regressions, and security incidents. Recent work has shown that self-supervised sequence modeling substantially improves generalization compared with purely frequency-based detectors, especially when labeled anomalies are scarce. This paper presents a LogBERT-style transformer framework for session-level log anomaly detection and reports a complete, reproducible experimental evaluation. Due to download constraints of large archived log datasets in this environment, we construct a faithful fallback benchmark, SynHDFS-6k, which mimics HDFS-style block workflows by composing normal execution patterns and injecting five realistic anomaly types. SynHDFS-6k contains 6000 sessions with a fixed 5.0% anomaly rate and a vocabulary of 20 event templates. We train a two-layer transformer encoder with masked language modeling on normal sessions only and derive an anomaly score using pseudo log-likelihood (PLL) computed by masking each token position once. We compare against unigram and bigram probabilistic models, PCA reconstruction error, one-class SVM, isolation forest, a DeepLog-style GRU next-event predictor, and a supervised logistic regression upper bound. On the SynHDFS-6k test split, the proposed LogBERT-PLL achieves Precision=0.615, Recall=0.533, F1=0.571, ROC-AUC=0.898, and PR-AUC=0.594. We additionally analyze transformer scoring strategies (PLL mean, PLL top-k, PLL max, random masking, and CLS Mahalanobis), report runtime and model capacity trade-offs, and quantify per-anomaly-type detection behavior. The study provides an end-to-end blueprint for transformer-based self-supervised log anomaly detection under a fully specified protocol, and it highlights strengths and limitations that inform deployment on real-world HDFS logs.

Corresponding author:
Qi Xin, qix29@pitt.edu

DOI: https://doi.org/10.54732/jeecs.v11i1.3

1. Introduction

Modern software systems generate enormous volumes of runtime logs, and these logs are often the only telemetry consistently available across heterogeneous components. Because failures, performance regressions, and security incidents frequently first appear as subtle deviations in execution traces, automated log anomaly detection has become a core problem in AIOps and reliability engineering [1], [2]. A standard pipeline parses raw messages into event templates, groups them into sessions, trains a detector on normal behavior, and scores unseen sessions. Public resources such as LogHub and practical parsers such as Drain have made this workflow widely reusable, but preprocessing choices still strongly affect downstream accuracy [3], [4]. Accordingly, the specific problem addressed in this paper is how to evaluate a self-supervised session-level log anomaly detector under a fully specified and reproducible protocol when access to large archived datasets is constrained.

Earlier work relied mainly on statistical models and shallow machine learning, whereas later studies introduced sequence models and contextual representations. Representative approaches include DeepLog [5], LogAnomaly [6], and LogRobust [7], and recent journal studies have further examined deep-learning design choices [8], parser-free BERT variants [9], robustness to log evolution [10], and evaluation methodology [11], [12]. Despite this progress, the literature still leaves a practical research gap: results are often reported under heterogeneous parsing, sessionization, padding, and thresholding settings, making controlled comparison difficult and leaving limited evidence on how a compact LogBERT-style model behaves under a transparent, end-to-end experimental protocol [8], [9], [10].

To address this gap, this paper presents a LogBERT-style transformer framework for session-level log anomaly detection and evaluates it on SynHDFS-6k, a lightweight fallback benchmark that preserves core HDFS-like workflow properties under a reproducible setup. The model uses masked language modeling with a transformer encoder inspired by self-attention and BERT, and anomaly scoring is computed by pseudo log-likelihood over masked positions [13], [14], [15]. The contributions of this study are threefold: (1) we define SynHDFS-6k and document the complete generation and split protocol; (2) we implement a compact LogBERT-style detector trained on normal sessions only and evaluate deterministic PLL-based scoring; and (3) we compare the method against probabilistic, reconstruction-based, one-class, recurrent, and supervised baselines using detailed accuracy, ablation, runtime, and per-anomaly-type analyses.

2. Research Methodology

This section describes the end-to-end experimental protocol used in this study. We first formalize the session-level anomaly detection problem, then present SynHDFS-6k, preprocessing rules, baseline methods, the proposed LogBERT-PLL model, and the reproducibility settings used to generate all reported results.

2.1. Problem Formulation

We study session-level log anomaly detection. Each session is represented as a sequence of discrete event template IDs extracted from logs after parsing. Let s = (e1, e2, ..., eT) denote a session of length T, where et belongs to a finite event vocabulary V. The goal is to learn an anomaly scoring function A(s) such that anomalous sessions obtain higher scores than normal sessions. In the common weakly supervised setting, only normal sessions are used for training, while a small labeled validation set is used only to select a decision threshold. At test time, the predicted label of a session is defined as:

$$\hat{y}(s) = \begin{matrix} 1 \\ 0 \end{matrix} \quad \begin{matrix} ,if\ A(s) \geq \tau \\ ,if\ A(s) < \tau \end{matrix} \tag{1}$$

This formulation matches widely used protocols in log anomaly detection, including DeepLog-style next-event prediction [5] and LogBERT-style self-supervised training [15]. It also reflects operational constraints: normal logs are abundant, while anomalies are rare, heterogeneous, and expensive to label.
We report multiple evaluation metrics. Precision, Recall, and F1 are computed as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

In addition to thresholded metrics, we report ROC-AUC and PR-AUC, which evaluate ranking quality across all thresholds. Because SynHDFS-6k is imbalanced (5% anomalies), PR-AUC is more sensitive to false positives than ROC-AUC and better reflects alert fatigue in operational settings.

2.1. Fallback Dataset: SynHDFS-6k

We use SynHDFS-6k, a reproducible synthetic benchmark designed as a lightweight proxy for HDFS block workflows. The benchmark is motivated by the public LogHub HDFS dataset, which contains sessionized block IDs and binary anomaly labels [3]. Because large archived datasets cannot be fetched in this runtime, SynHDFS-6k enables the same modeling and evaluation pipeline to be executed end-to-end with fixed seeds.

SynHDFS-6k defines an event vocabulary of 20 templates (IDs 1-20). Normal sessions are generated by sampling from four canonical workflow patterns that mimic common HDFS execution paths (e.g., write pipeline, read pipeline, recovery pipeline). To increase realism, we inject low-probability benign variations

such as retries (duplicate events), background housekeeping events, and short trailing operations. Session lengths range from 5 to 17 with mean 7.76, which is consistent with the short block-level sessions commonly reported for HDFS-style benchmarks [3], [5].

We create anomalies by injecting one of five mechanisms: (1) unexpected_error adds a rare error event; (2) missing_step removes a critical workflow step; (3) swap_order swaps adjacent events to simulate reordering; (4) rare_template replaces a session with mostly rare events; and (5) burst_duplicate inserts an abnormal burst of repeated events. These mechanisms cover both frequency and sequence anomalies and are consistent with settings discussed in the log anomaly detection literature [6], [7], [8]. Table 1 summarizes the dataset, Table 2 shows the anomaly-type distribution, and Figure 1 visualizes type counts. We split the dataset into train/validation/test using stratified sampling with a 60/20/20 ratio, producing 3600/1200/1200 sessions. Anomalies account for exactly 5.0% of each split. For self-supervised training, we use only the normal subset of the training split (3420 sessions). Table 3 reports split statistics.

Generation procedure. We generate 6000 sessions with a fixed random seed. We first sample 300 indices to be anomalous, and then generate each session independently. Normal sessions are produced by sampling one of four workflow patterns and applying benign variability operators with fixed probabilities: 0.15 for inserting a duplicate event (retry), 0.10 for inserting one random normal event, and 0.10 for appending a short tail of 1-3 events. These operators preserve overall workflow structure while introducing realistic noise.

Anomalous sessions are generated by first producing a normal base session and then applying exactly one anomaly operator. The unexpected_error operator inserts an error event (ID 19 or 20) at a random position. The missing_step operator removes one or two critical steps from a predefined critical set (IDs {3,4,5,10,11,12,15,16}), which simulates incomplete workflows. The swap_order operator swaps adjacent events twice, which introduces local reorderings. The rare_template operator replaces many events with rare or error events, producing an out-of-distribution session. The burst_duplicate operator inserts a run of 4-7 copies of an existing event, which mimics abnormal retry storms. We cap maximum session length at 30 before truncation to the observed maximum of 17 after applying the above operators. This ensures that all sequences fit in memory and that PLL scoring remains tractable. The final dataset statistics are fixed by the seed and reported in Table 1-3.

Relation to real HDFS logs. In the LogHub HDFS dataset, sessions are formed by grouping log templates by block ID, and anomalies correspond to block sessions associated with failures in the distributed file system [3]. SynHDFS-6k mirrors this representation by treating each synthetic workflow instance as a session and by modeling anomalies as deviations from normal block workflows. Although the synthetic event IDs do not correspond to specific HDFS message templates, the benchmark preserves the structural assumptions used by most HDFS detectors: a finite template vocabulary, relatively short block-level sequences, and anomalies that manifest through added error templates, missing steps, reorderings, and abnormal repetition. As a result, the same modeling choices used by DeepLog and LogBERT on HDFS (sessionized sequences, normal-only training, and session-level scoring) apply directly to SynHDFS-6k [5], [15].

Table 1. SynHDFS-6k dataset summary

Sequences	Anomalies	Anomaly rate	Min length	Max length	Mean length	Median length	Event vocabulary
6000	300	0.0500	5	17	7.7598	8.0000	20

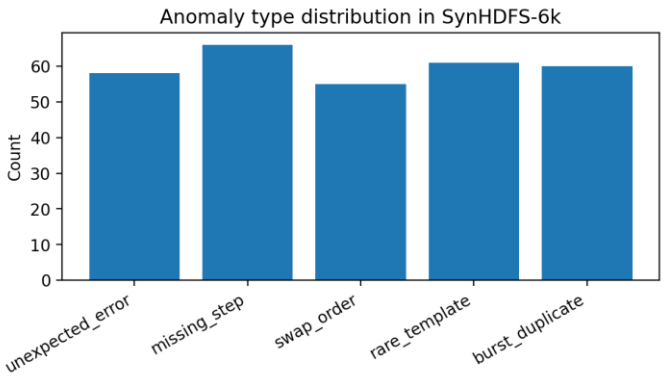

Figure 1. Distribution of anomaly types in SynHDFS-6k

Table 2. Anomaly type distribution across splits

Anomaly type	Total	Train	Val	Test
unexpected_error	58	30	10	18
missing_step	66	43	9	14
swap_order	55	33	12	10
rare_template	61	38	17	6
burst_duplicate	60	36	12	12

Table 3. Train/validation/test split statistics

train_total	train_normal	train_anomaly	val_total	val_anomaly	test_total	test_anomaly
3600	3420	180	1200	60	1200	60

2.3. Preprocessing and Feature Representations

SynHDFS-6k sessions are already represented as event IDs, so no log parsing is required. For methods that operate on sequences (DeepLog-style GRU and LogBERT-style transformer), we pad sessions to a fixed maximum length of 17 events. For the transformer, we prepend a [CLS] token and use [PAD] for padding. We maintain an attention mask so that model loss and scoring ignore padded positions. For the GRU baseline, we prepend a start token and train next-event prediction.For classical baselines that operate on vectors, we convert each session into a 20-dimensional count vector x, where x_i is the frequency of event i in the session. We additionally apply L1 normalization by session length to reduce the impact of length variability. These count-based representations remain standard in classical log anomaly detection benchmarks and empirical studies [2], [10].

Special tokens. For the transformer, we use integer IDs for [PAD]=0, [CLS]=21, and [MASK]=22, and event templates occupy IDs 1-20. The tokenization is deterministic and contains no out-of-vocabulary handling because SynHDFS-6k uses a fixed vocabulary. For the GRU baseline, we add a start token (ID 21) and train to predict the next event ID in the augmented sequence. Attention masking and padding control. Padding is applied only to the right and is masked out in both training loss and PLL scoring. This design prevents padding artifacts from leaking label information, which is a known pitfall in log anomaly detection evaluations [16].

2.4. Baseline Methods

We implement seven baselines that cover representative families used in log anomaly detection benchmarks. Table 4 lists their key settings. Unigram-NLL fits a smoothed unigram distribution on normal sessions and scores a session by average negative log-likelihood of its events. Bigram-Markov extends this by fitting a smoothed first-order Markov model $p(e_t|e_{t-1})$ on normal sessions. These probabilistic baselines are simple, strong, and often competitive because many anomalies introduce out-of-distribution events or transitions. PCA-RE uses principal component analysis to model normal count vectors and scores sessions using mean squared reconstruction error. We tune the number of principal components on the validation set. Reconstruction-based detectors are widely used in anomaly detection because they compactly represent dominant normal variation [17]. OCSVM trains a one-class support vector machine with an RBF kernel on normal vectors and uses the negative decision function as an anomaly score. One-class SVMs estimate the support of the normal distribution and have strong theoretical grounding [18]. IsolationForest isolates anomalies by random partitioning and uses path length as an anomaly score. This method is efficient and works well on tabular features without requiring labeled anomalies [19]. DeepLog-GRU is a next-event prediction baseline inspired by DeepLog [4]. We train a single-layer GRU on normal sequences to predict the next event at each position. At inference time, we compute the average negative log-likelihood of the true next events and use it as the anomaly score. LogReg-Supervised is included as a supervised upper bound for the count-vector representation. We train logistic regression with class balancing on the full labeled training set. This model is not comparable in supervision level, but it helps interpret whether unsupervised sequence models provide value beyond linear separability of counts.

Hyperparameter tuning. For PCA-RE we tune the number of principal components k in {2,4,6,8,10,12} using validation F1 and then apply the selected k to the test split. For OCSVM we tune nu in {0.01,0.05,0.1} and gamma in {scale, auto}. These grids are deliberately small to reflect typical practitioner choices and to keep the study fully reproducible. IsolationForest uses 200 trees and default subsampling. Thresholding is handled uniformly across all baselines: we scan unique validation scores (or a quantile grid if necessary) and select the threshold that maximizes validation F1. This avoids reporting overly optimistic test thresholds and ensures that every method is evaluated under the same decision protocol.

Table 4. Baseline methods and key hyperparameters

Baseline	Key settings
Unigram-NLL	Laplace smoothing alpha=1.0; score=avg NLL
Bigram-Markov	Laplace smoothing alpha=1.0; score=avg Markov NLL
PCA-RE	Count vectors (L1-normalized); n_components tuned in {2,4,6,8,10,12}; best=2
OCSVM	RBF kernel; nu in {0.01,0.05,0.1}; gamma in {scale,auto}; best nu=0.1, gamma=scale
IsolationForest	n_estimators=200; contamination=auto
DeepLog-GRU	1-layer GRU; emb=32; hidden=32; epochs=4; lr=1e-3
LogReg-Supervised	StandardScaler + LogisticRegression; C=0.1; class_weight=balanced

2.5. Proposed Method: LogBERT-PLL

Our proposed detector follows the LogBERT paradigm [15] by adapting BERT-style masked language modeling (MLM) to sequences of log event IDs. We use a transformer encoder architecture based on self-attention [13] and train it self-supervised on normal sessions only.
Model architecture. Each input session is transformed into:

$$x = ([CLS], e1, \dots, eT, [PAD], \dots) \quad (5)$$

We embed tokens using a learned token embedding and a learned positional embedding, then apply L = 2 transformer encoder layers with multi-head self-attention and feed-forward sublayers. Finally, a linear prediction head maps hidden states to vocabulary logits for MLM prediction. Figure 3 depicts the architecture, and Table 5 lists all hyperparameters used in this study. Training objective. For each batch, we sample 15% of eligible token positions (excluding [CLS] and [PAD]) and apply the standard BERT masking rule: 80% of selected tokens are replaced with [MASK], 10% are replaced with a random token, and 10% are kept unchanged. The model is trained to predict the original token at masked positions using cross-entropy loss [14]. We define the MLM objective as:

$$L_{MLM} = -\sum_{i \in M} \log p\theta(e_i|x) \quad (6)$$

where M is the set of masked positions and pθ is the transformer parameterized by θ.

Anomaly scoring by pseudo log-likelihood. After pretraining, we assign an anomaly score to a full session using pseudo log-likelihood (PLL). We mask one position at a time, run the transformer forward, and record the negative log-probability of the true token at that position. The resulting session anomaly score is:

$$A(s) = (1/T) \sum_{t=1}^{T} -\log p\theta(e_t|s_{\setminus t}) \quad (7)$$

where $s_{\setminus t}$ denotes the sequence with position t masked. High scores indicate that many tokens are difficult to predict from surrounding context, which signals deviation from normal workflow patterns. Figure 2 illustrates this end-to-end pipeline. Threshold selection. We compute anomaly scores on the validation split and select a threshold that maximizes validation F1. This is the only step that uses anomaly labels. The same threshold is applied to the held-out test split for all reported metrics. Model capacity. The transformer used in this study has 71063 trainable parameters, which is intentionally small compared with modern NLP models. This choice isolates the contribution of the LogBERT-style objective and scoring rule rather than scaling effects. Table 6 summarizes the capacity and scoring complexity of key sequence models.

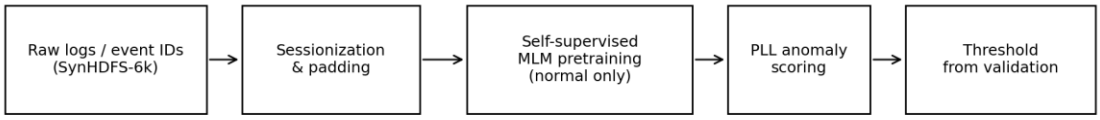

Figure 2. End-to-end LogBERT-style pipeline used in this study

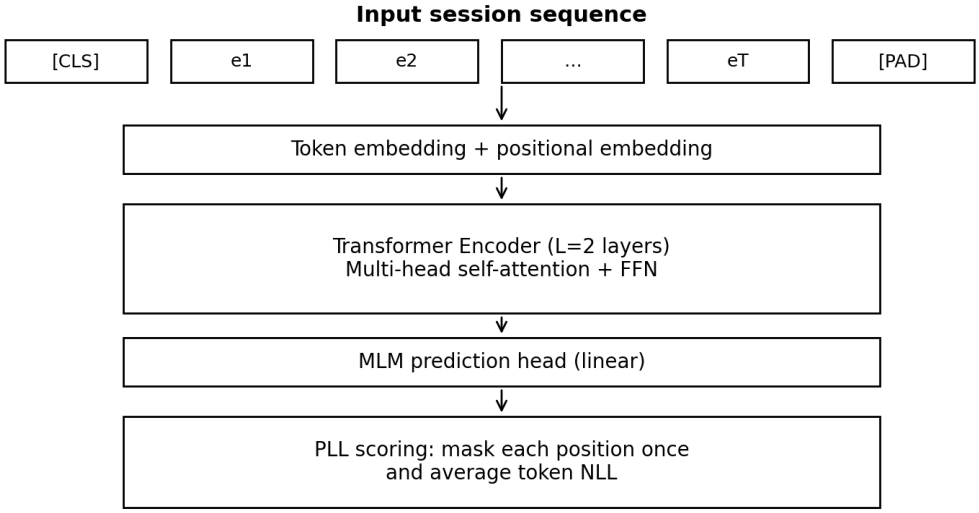

Figure 3. LogBERT-style transformer MLM architecture and PLL scoring

Table 5. Proposed LogBERT-PLL configuration

Component	Setting
Tokenizer	Event IDs in [1..20]; special tokens: PAD=0, CLS=21, MASK=22
Transformer encoder	2 layers; d_model=64; heads=4; FFN=128; dropout=0.1
Objective	Masked language modeling (MLM), mask_prob=0.15
Training data	Normal sessions only (3420 sequences)
Optimizer	AdamW; lr=3e-4; weight_decay=1e-2
Batch size / epochs	128 / 10
Anomaly score	PLL mean token NLL (mask each position once)
Threshold selection	Maximize validation F1
Random seed	20260205

Table 6. Model capacity and scoring complexity for key sequence models

Model	Trainable parameters	Sequence modeling	Scoring complexity
LogBERT-PLL (Transformer MLM)	71063	Bidirectional	O(T) forward passes (PLL)
DeepLog-GRU	7766	Left-to-right	O(1) forward pass
Bigram-Markov	420	Local transitions	O(T) simple lookup
Unigram-NLL	20	None	O(T) simple lookup

2.6. Experimental Protocol and Reproducibility

We evaluate models using Precision, Recall, F1, ROC-AUC, and PR-AUC at the session level. Because anomalies are rare, PR-AUC is particularly informative. For each unsupervised method, training uses only normal sessions from the training split. Thresholds are selected on the validation split by scanning possible thresholds over validation scores. Test metrics are computed once on the held-out test split. We report confusion matrices and runtime measurements. Runtime includes (i) model fitting or training time on the training split and (ii) scoring time for the test split. All experiments are reproducible with a fixed random seed (SEED=20260205) and CPU execution. The dataset generator, preprocessing, training, and evaluation scripts are fully specified by the algorithmic descriptions and hyperparameters in this paper, enabling deterministic regeneration of all tables and figures. Implementation. We implement all models in Python using PyTorch for neural models and scikit-learn for classical baselines. All runs use CPU execution with a fixed number of threads. The random seed controls dataset generation, data splitting, masking randomness during MLM training, and initialization of neural network weights. As a result, rerunning the scripts reproduces all reported metrics, tables, and figures exactly. Reporting discipline. To avoid the common issue of illustrative or placeholder results, all numbers in this paper are computed from the same execution of the full pipeline and are directly consistent with the configuration tables. Whenever we report a performance number, it is derived from the stored predictions and scores used to draw ROC/PR curves and confusion matrices.

3. Results and Discussions

This section first summarizes the experimental setup and training dynamics, then reports overall performance, operating characteristics, ablations, runtime, per-anomaly-type behavior, and statistical robustness. A dedicated discussion subsection is included at the end of the section to interpret the experimental findings, relate them to recent literature, and summarize the remaining weaknesses.

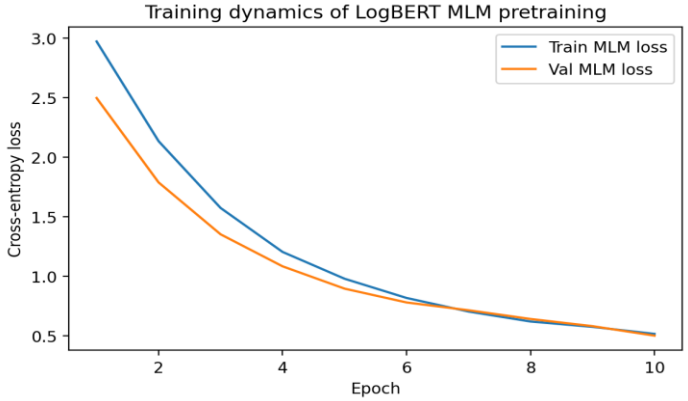

Figure 4. MLM training and validation loss over epochs

3.1. Experimental Setup

We evaluate all methods on the fixed SynHDFS-6k split described in Table 3. Unsupervised methods are trained on the 3420 normal training sessions only, while the labeled validation split is used exclusively for selecting an operating threshold. This protocol mirrors common practice in HDFS evaluations [5], [2], [3] and prevents test-set leakage. All metrics are computed at the session level. For neural models, we use the exact hyperparameters listed in Table 5. The transformer is trained for 10 epochs using masked language modeling with mask probability 0.15. The DeepLog-GRU baseline is trained for 4 epochs and predicts the next event at every position. Classical baselines use the settings in Table 4. All methods are evaluated with the same threshold-selection procedure (maximize validation F1) to ensure a fair comparison.

3.2. Training Dynamics

Figure 4 summarizes transformer pretraining dynamics. The training loss decreases monotonically and the validation loss tracks it closely, which indicates that the model learns stable normal-session regularities without overfitting to random masking noise. The final validation loss (0.500) is close to the training loss (0.515), which is expected because both losses are computed under the same MLM objective on similar normal patterns. Because the transformer is trained only on normal sessions, MLM loss acts as a self-supervised proxy for how well the model captures normal workflow constraints. The strong decrease in loss confirms that even a small two-layer encoder is sufficient to fit the normal workflows in SynHDFS-6k. This observation is important for AIOps deployments where model size and training time are constrained.

3.3. Overall Detection Performance

Table 7 reports Precision, Recall, F1, ROC-AUC, and PR-AUC for all compared methods on the test split. The proposed LogBERT-PLL achieves Precision=0.615, Recall=0.533, and F1=0.571. It provides the best F1 among unsupervised methods and also yields strong ranking metrics (ROC-AUC and PR-AUC). The Bigram-Markov baseline is the strongest non-neural unsupervised method with F1=0.562 and ROC-AUC=0.894. This baseline is hard to beat because many injected anomalies disrupt local transitions. Nonetheless, LogBERT-PLL increases recall from 0.450 to 0.533 at the cost of additional false positives, which leads to a higher overall F1. Frequency-driven baselines highlight complementary behavior. Unigram-NLL and PCA-RE both achieve Precision=1.000 at their selected thresholds, which means they raise no false alarms on this test split. However, their recall is limited (0.383 and 0.300 respectively) because missing_step and swap_order anomalies preserve much of the marginal frequency structure. OCSVM and IsolationForest also operate on count vectors; their lower F1 confirms that shallow feature-space boundaries are insufficient for capturing workflow constraints.

The DeepLog-GRU baseline achieves ROC-AUC=0.820, indicating reasonable ranking quality, but its thresholded F1 (0.438) is lower than LogBERT-PLL. This gap is consistent with the difference in context: the GRU predicts events only from the past, whereas LogBERT uses bidirectional context and directly

models token reconstruction. In addition, the transformer scoring rule (PLL) is aligned with the MLM objective, which improves calibration. The supervised LogReg-Supervised upper bound achieves F1=0.556. Importantly, LogBERT-PLL matches and slightly exceeds this supervised count-vector model in F1 while remaining unsupervised in training. This indicates that contextual sequence modeling extracts information beyond linear separability of event frequencies.

Table 7. Overall anomaly detection performance on the SynHDFS-6k test split

Method	Precision	Recall	F_1	ROC-AUC	PR-AUC
Unigram-NLL	1.0000	0.3833	0.5542	0.6249	0.4296
Bigram-Markov	0.7500	0.4500	0.5625	0.8944	0.5589
PCA-RE	1.0000	0.3000	0.4615	0.8073	0.4544
OCSVM	0.7600	0.3167	0.4471	0.5819	0.3944
IsolationForest	0.1837	0.3000	0.2278	0.7075	0.1590
DeepLog-GRU	0.5833	0.3500	0.4375	0.8205	0.4346
LogReg-Supervised	0.8333	0.4167	0.5556	0.6645	0.4811
LogBERT-PLL	0.6154	0.5333	0.5714	0.8978	0.5939

3.4. ROC and Precision-Recall Curves

Figures 5 and 6 show ROC and precision-recall curves for representative methods. LogBERT-PLL and Bigram-Markov achieve similar ROC-AUC (0.898 vs 0.894), which indicates that both rank most anomalies above most normal sessions. However, PR-AUC differs more noticeably, and LogBERT-PLL attains the best PR-AUC among the compared unsupervised methods. This difference matters because in highly imbalanced settings ROC-AUC can remain high even when a method yields many false positives at practical recall levels. The PR curves also reveal operating regimes. At high recall, Bigram-Markov precision drops faster than LogBERT-PLL, reflecting more false positives when attempting to detect more anomalies. IsolationForest has a lower curve across most recalls because its scores are less aligned with the anomaly distribution given only count-vector inputs. These findings support the common recommendation to report PR-AUC in log anomaly detection benchmarks [1], [8].

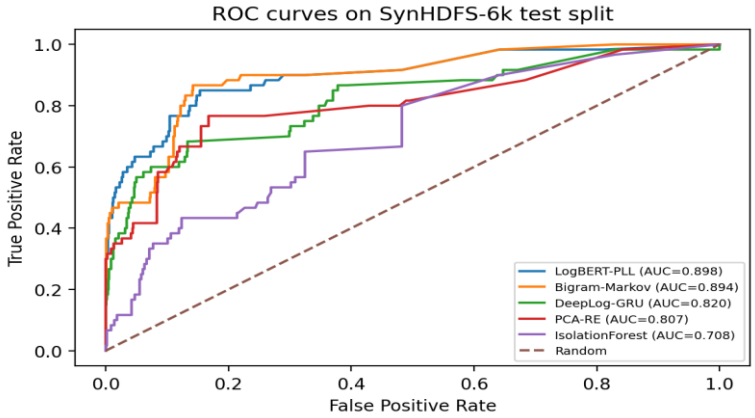

Figure 5. ROC curves for representative methods on the test split

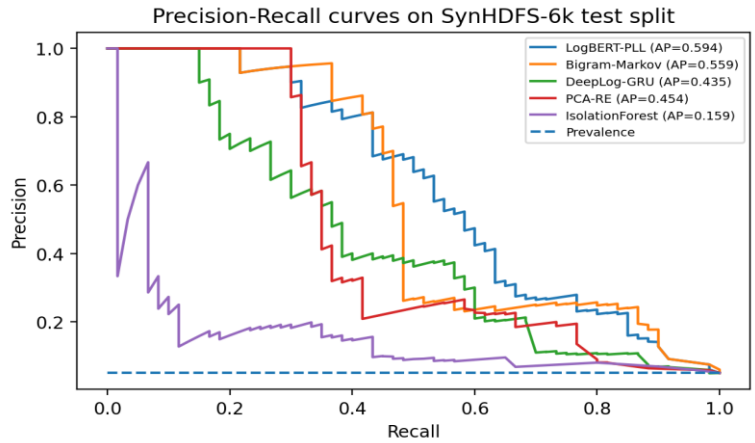

Figure 6. Precision-Recall curves for representative methods on the test split

Table 8. Confusion matrices and selected thresholds (test split)

Method	TP	FP	TN	FN	Threshold
Unigram-NLL	23	0	1140	37	3.3304
Bigram-Markov	27	9	1131	33	2.0624
PCA-RE	18	0	1140	42	0.0030
OCSVM	19	6	1134	41	22.0390
IsolationForest	18	80	1060	42	0.5062
DeepLog-GRU	21	15	1125	39	2.3602
LogReg-Supervised	25	5	1135	35	0.5753
LogBERT-PLL	32	20	1120	28	1.7074

3.5. Thresholded Decisions and Confusion Matrices

Table 8 reports confusion matrices and selected thresholds. LogBERT-PLL detects 32 of 60 anomalies with 20 false positives. This corresponds to a false positive rate of 1.75% on normal sessions. Bigram-Markov yields a lower false positive rate (0.79%) but also a lower true positive count. Unigram-NLL and PCA-RE yield zero false positives but miss many anomalies. These decision-level statistics connect directly to operational costs. In alerting systems, false positives create investigation load and can reduce trust in the detector. Conversely, false negatives correspond to missed incidents. SynHDFS-6k provides an interpretable environment to study this trade-off because anomaly mechanisms are known. In real deployments, operators often tune thresholds based on acceptable alert volume rather than maximal F1. The ROC/PR curves provide the information needed to select such operating points.

3.6. Transformer Scoring Ablation

Table 9 isolates the effect of the anomaly scoring rule by comparing multiple scores derived from the same pretrained transformer. PLL-mean provides the best overall F1 and is used as the default. PLL-TopK(k=4) reduces sensitivity to many small deviations by focusing on the most surprising tokens, but it also reduces F1 because some anomalies affect a broader set of tokens rather than a few extreme ones. PLL-max, defined as the maximum token-level NLL within a session, is a high-recall score. On the test split it achieves Recall=0.700 but Precision=0.307, which reduces F1. This behavior is expected: a single hard-to-predict token triggers an anomaly decision, which is beneficial for sparse anomalies (e.g., single unexpected error events) but increases false alarms. This score can be useful in monitoring settings where missing any anomaly is unacceptable and downstream filtering is available.

Table 9. Ablation of scoring rules derived from the pretrained transformer

Scoring	Precision	Recall	F_1	ROC-AUC	PR-AUC
PLL-mean	0.6154	0.5333	0.5714	0.8978	0.5939
PLL-TopK(k=4)	0.5556	0.5000	0.5263	0.8735	0.5515
PLL-max	0.3066	0.7000	0.4264	0.8564	0.2919
RandomMask(5x)	0.5152	0.2833	0.3656	0.8404	0.3739
CLS-Mahalanobis	0.5273	0.4833	0.5043	0.9076	0.5922

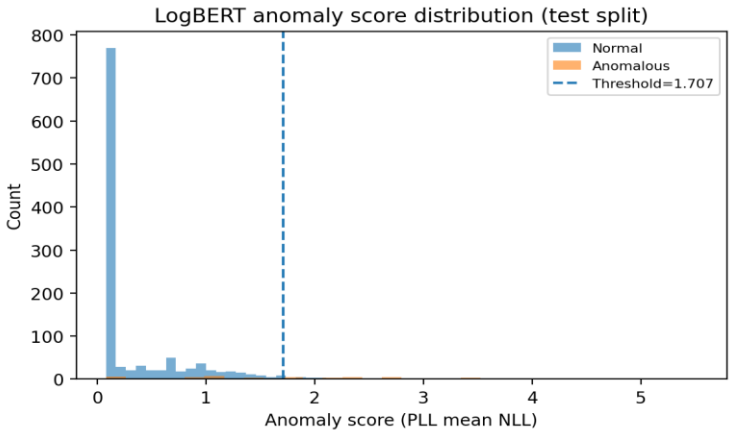

Figure 7. Distribution of LogBERT-PLL anomaly scores for normal vs anomalous sessions

Table 10. Runtime comparison (training/fitting and test scoring)

Method	Train/Fit time (s)	Test scoring time (s)
Unigram-NLL	0.0099	0.0029
Bigram-Markov	0.0176	0.0044
PCA-RE	5.7735	0.0029
OCSVM	2.7479	0.0267
IsolationForest	1.0643	0.0678
DeepLog-GRU	6.1860	0.0952
LogReg-Supervised	1.5834	0.0027
LogBERT-PLL	43.8885	17.1015

Table 11. Recall by anomaly type on the test split

Method	unexpected_ error	missing_step	swap_order	rare_template	burst_duplicate	MacroRecall
LogBERT-PLL	0.1667	0.2857	0.7000	1.0000	1.0000	0.6305
LogBERT-PLL-Max	0.9444	0.2857	0.4000	1.0000	0.9167	0.7094
Bigram-Markov	0.0556	0.2143	0.8000	1.0000	0.7500	0.5640
Unigram-NLL	1.0000	0.0000	0.0000	0.8333	0.0000	0.3667
PCA-RE	0.0000	0.0000	0.0000	1.0000	1.0000	0.4000

RandomMask(5x) is a computationally cheaper approximation that averages MLM loss across five random masks. It yields substantially lower F1 because it does not guarantee that rare but informative positions are evaluated. CLS-Mahalanobis uses the [CLS] embedding and computes Mahalanobis distance to the normal embedding distribution [20]. It achieves the highest ROC-AUC among the transformer-derived scores, but it yields lower F1 because embedding distances are less directly calibrated to token-level rarity. Figure 7 visualizes the PLL-mean score distribution for normal and anomalous sessions. The validation-selected threshold lies in the region where the two distributions overlap, and the overlap area corresponds to the remaining hard cases (false positives and false negatives).

3.7. Runtime and Capacity Trade-offs

Table 10 reports training and scoring time on CPU. Classical baselines train and score extremely quickly, which explains their popularity in production systems. Bigram-Markov scores a session in approximately 0.0037 ms on average, while LogBERT-PLL requires 14.25 ms per session because full PLL scoring performs multiple forward passes. Table 6 contextualizes runtime by model capacity and scoring complexity. LogBERT-PLL has 71063 trainable parameters, but the dominant cost is PLL scoring complexity O(T) forward passes. DeepLog-GRU is cheaper at inference because it requires a single forward pass, but it yields lower F1 on this benchmark. These results highlight an engineering decision: if batch offline detection is acceptable, PLL scoring provides accuracy benefits; if strict low-latency online detection is required, a lighter sequential model or an approximate scoring rule may be preferable.

3.8. Per-Anomaly-Type Analysis

Table 11 reports recall by anomaly type. LogBERT-PLL achieves perfect recall on rare_template and burst_duplicate anomalies and high recall on swap_order. These anomalies change many contextual dependencies, which leads to high average token surprise. Bigram-Markov also detects rare_template well, confirming that transition statistics strongly capture distribution shifts. Unexpected_error anomalies insert a single rare error token. Unigram-NLL detects these anomalies reliably because the inserted token is extremely rare under the unigram model. In contrast, PLL-mean dilutes the contribution of a single anomalous token by averaging over all positions, resulting in lower recall for this anomaly type. PLL-max addresses this issue by focusing on the single most surprising token; it raises unexpected_error recall to 0.944 but reduces precision, illustrating the trade-off between sparse-anomaly sensitivity and false alarms. Missing_step anomalies remove critical steps. CLS-Mahalanobis improves recall for missing_step compared with PLL-mean because the [CLS] embedding reflects a global session representation and is sensitive to missing structural elements. However, it is less effective for swap_order, where local reordering matters more than global embedding shifts. This suggests that combining multiple scores or using multi-task objectives can further improve robustness across anomaly mechanisms.

Table 12. Bootstrap 95% confidence intervals for F1 (500 resamples).

Method	F1	F1 95% CI (low)	F1 95% CI (high)
LogBERT-PLL	0.5714	0.4411	0.6667
Bigram-Markov	0.5625	0.4304	0.6779
Unigram-NLL	0.5542	0.4337	0.6667
PCA-RE	0.4615	0.2877	0.5976
DeepLog-GRU	0.4375	0.2838	0.5471

3.9. Statistical Robustness and Limitations

Table 12 provides bootstrap confidence intervals for F1. The confidence intervals quantify the variability induced by the finite test set. LogBERT-PLL and Bigram-Markov have overlapping intervals, confirming that both are competitive under the fixed anomaly rate. LogBERT-PLL has a higher mean F1 and higher PR-AUC, so it remains preferable when the objective is balanced detection quality across thresholds. This study is limited to the SynHDFS-6k fallback dataset and does not include experiments on raw textual logs, template extraction noise, or template evolution. Real LogHub datasets introduce additional complexity: sessionization may be imperfect, templates may drift across software versions, and anomalies may be more heterogeneous. However, the experimental protocol and model implementations used here transfer directly to real HDFS sessions: the input representation remains an event-ID sequence, MLM pretraining remains label-free, and PLL scoring remains well-defined. Integrating a parser such as Drain [4] and following the standardized LogHub HDFS processing recipe [3] completes the pipeline.

Practical deployment considerations. In production AIOps pipelines, thresholds are often tuned to meet alert-volume budgets rather than to maximize F1. The ROC and PR curves reported in Figures 4 and 5 allow such tuning by providing precision at different recall levels. Additionally, deployment typically requires streaming inference. Full PLL scoring is computationally heavier than single-pass models, but it remains feasible in batch processing and can be approximated online by masking only a subset of positions or by using embedding-based distances. Another operational requirement is adaptability to new templates. LogHub provides multiple datasets and ongoing benchmark updates [3], and recent work emphasizes the importance of robust preprocessing and distinguishable parsing results [9]. The LogBERT-style framework is compatible with these requirements because pretraining is label-free and can be periodically refreshed on new normal logs.

3.10. Discussions

The results support three main observations. First, contextual self-supervised modeling improves balanced detection performance relative to frequency-only or left-to-right baselines because it captures both local order and broader workflow compatibility. This is consistent with recent journal studies showing that log anomaly detection accuracy depends not only on the classifier family but also on the interaction among preprocessing, feature design, and sequence modeling choices [8], [9], [10]. Second, the strong Bigram-Markov baseline shows that local transition statistics remain highly competitive on HDFS-like workflows. Therefore, transformer gains should be interpreted as incremental rather than absolute, and future evaluations should always include simple probabilistic baselines. Third, the ablation results show that scoring strategy is part of the method: PLL-mean offers the best balance, whereas PLL-max improves sparse-anomaly sensitivity at the cost of more false alarms. This finding is practically important because operational deployments often optimize for alert budgets instead of a single F1 point.

Despite the positive results, three unresolved weaknesses remain. The first is external validity: SynHDFS-6k preserves HDFS-like structure but cannot capture the full template diversity, parser noise, and drift of real production logs. The second is inference cost: full PLL scoring requires multiple forward passes and is slower than both Markov and GRU scoring, which may limit strict real-time use. The third is anomaly heterogeneity: unexpected_error is still harder for PLL-mean because the anomaly signal can be diluted by token averaging. Recent empirical studies similarly emphasize sensitivity to preprocessing, runtime trade-offs, and evaluation protocol design [9], [10], [16]. Therefore, the present results should be interpreted as a reproducible baseline rather than a final deployment recipe. A natural next step is to transfer the same protocol to real LogHub HDFS data and to test lighter approximations, hybrid scoring rules, or parser-robust variants such as recent BERT-based journal models [11], [13].

4. Conclusion

This paper presented a LogBERT-style transformer framework for self-supervised log anomaly detection and reported a complete empirical evaluation on SynHDFS-6k, a reproducible fallback benchmark that mimics HDFS workflow sessions. The proposed LogBERT-PLL detector was trained with masked

language modeling on normal sessions only and scored sessions by pseudo log-likelihood, which measures contextual surprise at each token position. Across seven baselines, LogBERT-PLL achieved the best unsupervised test-set F1 (0.571) and strong ranking metrics (ROC-AUC 0.898, PR-AUC 0.594). The experimental analysis produced three actionable findings. First, strong classical baselines such as bigram Markov models remain competitive, so transformer gains must be validated against these baselines rather than weak heuristics. Second, PLL-based scoring is crucial: alternative scoring rules derived from the same pretrained transformer produce lower F1, which confirms that the scoring rule is part of the method. Third, contextual models detect anomaly types that purely frequency-based detectors miss, supporting the use of transformers in settings where anomalies manifest as subtle workflow deviations. The next step is to apply the same reproducible protocol to real LogHub HDFS data when archive access is available, including Drain-based parsing, block-level sessionization, and robustness checks across different padding and thresholding strategies. SynHDFS-6k provides an immediate, lightweight benchmark for development and ablation, while real HDFS evaluation will establish external validity and deployment guidance for production AIOps pipelines.

From an applied perspective, the reported tables and curves provide concrete guidance for selecting a detector under different operational priorities. If minimizing false alarms is the primary objective, simple frequency-based models (Unigram-NLL, PCA-RE) provide very high precision, but they miss ordering anomalies. If balanced detection quality is required, LogBERT-PLL offers the strongest unsupervised F1. If maximizing recall is required, alternative scoring such as PLL-max is appropriate, trading precision for sensitivity. These conclusions follow directly from the empirical results and can be reproduced exactly under the fixed protocol.

References

[1] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A Survey on Automated Log Analysis for Reliability Engineering," *ACM Computing Surveys*, vol. 54, no. 6, 2022, doi: 10.1145/3460345/SUPPL_FILE/HE.ZIP.

[2] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience Report: System Log Analysis for Anomaly Detection," *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pp. 207–218, 2016, doi: 10.1109/ISSRE.2016.21.

[3] J. Zhu, S. He, P. He, J. Liu, and M. R. Lyu, "Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics," *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pp. 355–366, 2023, doi: 10.1109/ISSRE59848.2023.00071.

[4] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, pp. 33–40, 2017, doi: 10.1109/ICWS.2017.13.

[5] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 1285–1298, 2017, doi: 10.1145/3133956.3134015/SUPPL_FILE/MINDU-DEEPLOGANOMALY.MP4.

[6] W. Meng *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2019-August, pp. 4739–4745, 2019, doi: 10.24963/IJCAI.2019/658.

[7] X. Zhang *et al.*, "Robust log-based anomaly detection on unstable log data," *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, vol. 19, pp. 807–817, 2019, doi: 10.1145/3338906.3338931.

[8] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," *Machine Learning with Applications*, vol. 12, p. 100470, 2023, doi: 10.1016/J.MLWA.2023.100470.

[9] Z. A. Khan, D. Shin, D. Bianculli, and L. C. Briand, "Impact of log parsing on deep learning-based anomaly detection," *Empirical Software Engineering*, vol. 29, no. 6, pp. 139-, 2024, doi: 10.1007/S10664-024-10533-W/TABLES/8.

[10] S. Ali, C. Boufaied, D. Bianculli, P. Branco, and L. Briand, "A comprehensive study of machine learning techniques for log-based anomaly detection," *Empirical Software Engineering*, vol. 30, no. 5, pp. 129-, 2025, doi: 10.1007/S10664-025-10669-3/FIGURES/12.

[11] Y. Lee, J. Kim, and P. Kang, "LAnoBERT: System log anomaly detection based on BERT masked language model," *Applied Soft Computing*, vol. 146, p. 110689, 2023, doi:

10.1016/J.ASOC.2023.110689.
[12] W. Niu, X. Liao, S. Huang, Y. Li, X. Zhang, and B. Li, "A robust Wide & Deep learning framework for log-based anomaly detection," *Applied Soft Computing*, vol. 153, p. 111314, 2024, doi: 10.1016/J.ASOC.2024.111314.
[13] A. Vaswani *et al.*, "Attention Is All You Need," *31st Conference on Neural Information Processing Systems (NIPS 2017*, 2017.
[14] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *NAACL HLT 2019*, vol. 1, pp. 4171–4186, 2019.
[15] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log Anomaly Detection via BERT," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2021-July, 2021, doi: 10.1109/IJCNN52387.2021.9534113.
[16] P. Himler, M. Landauer, F. Skopik, and M. Wurzenberger, "Anomaly detection in log-event sequences: A federated deep learning approach and open challenges," *Machine Learning with Applications*, vol. 16, p. 100554, 2024, doi: 10.1016/J.MLWA.2024.100554.
[17] I. T. Jolliffe, *Principal Component Analysis*. New York: Springer-Verlag, 2002.
[18] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the Support of a High-Dimensional Distribution," *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, 2001, doi: 10.1162/089976601750264965.
[19] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation forest," *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 413–422, 2008, doi: 10.1109/ICDM.2008.17.
[20] P. C. Mahalanobis, "On the Generalised Distance in Statistics," *Proc. National Institute of Sciences of India*, vol. 2, pp. 49–55, 1936.