Anomaly Detection in Log Data: A Comparative Study

Ondřej Sedláček
CESNET
Prague, Czech Republic
ORCID: 0009-0004-8670-7396

Martin Žádník
CESNET
Prague, Czech Republic
ORCID: 0000-0002-2099-2348

Václav Bartoš
CESNET
Prague, Czech Republic
ORCID: 0000-0003-4334-3559

***Abstract*—Log data are valuable for monitoring system health and security, but inconsistent preprocessing across studies makes it hard to compare anomaly detection methods. We present a unified framework that standardizes preprocessing through template extraction, sequence grouping, and feature encoding, enabling fair comparisons. The detectors are evaluated on three benchmarks (HDFS, BGL, Thunderbird) using the Drain parser, multiple representation modules, and both classical and modern models with tuned hyperparameters. Our results show that transformer-based methods perform well on randomized data but degrade under sequential evaluation, while classical models are more robust. We also show that minor preprocessing choices (e.g., template merging or sequence length) can shift F_1 scores by over 10%. We release our framework as an open-source testbench for reproducible research.**

I. Introduction

Detecting anomalies in system logs is vital for the reliability, availability, and security of modern infrastructure, as logs often provide the first and sometimes only evidence of failures and serve as the main forensic trace in post-incident reviews. Log-based anomaly detection enables early identification of latent issues, such as rising error rates or resource contention, before they cause outages. Treating logs as proactive sensor data accelerates root cause analysis, exposes security risks (e.g., bursts of authentication failures or irregular access patterns), and improves cost management, preserving user experience by addressing minor anomalies before they escalate.

Identifying anomalous behavior in log data is fundamentally challenging. Log formats are heterogeneous, often semi-structured or unstructured, with content varying across systems. Preprocessing steps such as parsing logs into templates or grouping sequences are nontrivial and differ across studies. Anomaly definitions also vary by task, from intrusion detection to system failure prediction, further complicating design and evaluation of methods. Evaluations often rely on unpublished pipelines [1]–[6], making direct comparisons difficult. With few alternatives, many authors instead compare results directly to prior papers despite their differences [7], [8].

Many benchmarking efforts evaluate individual methods, but not all authors publish code for replication. As a result, researchers often rely on reimplementations or reconstruct models from descriptions. For example, most works reimplemented DeepLog [6] due to the lack of an official release. Studies also use different template extraction rules or sequence grouping strategies, which can obscure a method's true performance. More reproducible and comparable evaluations require acknowledging and standardizing these practical aspects.

In this work, we introduce a unified, extensible evaluation framework for log-anomaly detection[1]. The pipeline covers the full lifecycle: parsing logs with the Drain parser [9], structuring sequences with consistent grouping strategies, extracting features via count-based and semantic representations, and evaluating detectors from classical statistical models to modern deep learning. All components are modular, enabling easy swapping of detectors, representations, and datasets while ensuring consistent preprocessing, tuning, and metrics.

We use the framework to study three research questions:

- **RQ1:** How do popular log-anomaly methods compare in precision, recall, and F_1 under a unified protocol?
- **RQ2:** How do preprocessing choices, such as dataset filtering or sequence grouping, affect performance?
- **RQ3:** How does training on shuffled logs compare to training on temporally continuous logs?

To answer these, we evaluate nine methods on three datasets (HDFS, BGL, Thunderbird), covering diverse systems, log structures, and granularities. All experiments use identical conditions with controlled seeds, Optuna-tuned [10] hyperparameters, and standardized metrics.

We summarize our contributions as follows:

- **Propose** a modular, end-to-end evaluation pipeline that unifies parsing, sequence grouping, representation, model integration, and evaluation into one open-source framework (section III).
- **Benchmark** nine methods on three standard datasets (section IV), showing transformer models perform best under randomized splits but degrade under sequential evaluation, and that preprocessing choices can shift F_1 by over 10%.
- **Analyze** data-size sensitivity (section IV-B) by evaluating methods at 10% and 1% training ratios under shuffled and sequential splits, revealing the impact of limited data and temporal continuity.

[1]https://github.com/xsedla1o/LogADComp

978-3-903176-75-1

II. Related work

Most automated log-anomaly detection methods rely on extracting log templates during parsing. Approaches include clustering-based parsers such as SLCT [11] and IPLoM [12], subsequence-based parsers like Spell [13], tree heuristics such as Drain [9], and neural parsers including NuLog [14] and SwissLog [15]. Clustering and subsequence parsers can oversegment templates when parameters have few unique values, while tree and neural methods mitigate this with rules or learned representations. Still, even tree-based and neural parsers may misclassify semantically important tokens as dynamic parameters [9], [16].

After parsing, log templates are converted into compact representations. Classical event-count vectors (ECVs) record template frequencies within a sequence, offering interpretability but discarding order and semantics [17]. Word-embedding representations (e.g., Word2Vec, FastText) capture semantic relationships between log tokens, aiding generalization to unseen events [4]. Transformer-based contextual embeddings represent tokens in context, yielding richer representations that capture subtle variations in log messages [1], [7].

Early log-anomaly detectors relied on rule-based systems and invariant mining to encode domain knowledge. Tools such as Logsurfer [18] and SEC [19] use expert-defined patterns and real-time rules, while invariant mining discovers execution invariants from ECVs to detect violations of expected behavior [20]. These approaches require extensive manual tuning and often miss anomalies outside predefined rules.

Unsupervised machine-learning methods reduced manual effort by modeling normal behavior from unlabeled logs. Principal Component Analysis on ECVs flags anomalies via reconstruction error [17], while clustering with TF–IDF weighting groups similar sequences to isolate outliers [21]. These methods adapt to changing log distributions but struggle with high dimensionality and limited semantic context.

Semi- and supervised baselines improved detection by using labeled data to separate normal and abnormal patterns. One-class SVMs trained on normal logs define anomaly boundaries [7], [22], while decision-tree and SVM classifiers applied to template-index and TF–IDF features achieve high precision when anomaly samples are available [23], [24]. These methods, however, rely on labeled anomalies covering all fault types and may fail to generalize to unseen events.

With deep learning, sequence models and neural embeddings have become central to log-anomaly detection. DeepLog [6] trains an LSTM on template-ID sequences to predict the next event, flagging anomalies when the true event is outside the top predictions. LogAnomaly [4] creates Template2Vec embeddings from GloVe vectors and applies an LSTM over event windows for streaming detection. RobustLog [5] uses a bidirectional LSTM with TF–IDF–weighted FastText embeddings and attention to classify log sequences, improving robustness to template variation. NeuralLog [16] employs WordPiece tokenization with BERT embeddings in a transformer encoder for binary classification, avoiding template parser errors. LogBERT [7] combines masked language modeling with hypersphere minimization to cluster normal logs and detect anomalies via masked-token prediction errors. LAnoBERT [1] fine-tunes BERT to predict each token in a log, aggregates token-level losses and top-k probabilities into anomaly scores, and adds caching to accelerate inference.

However, fair and reproducible evaluation remains difficult, as practices are fragmented along several axes. First, while standard datasets (e.g., HDFS, BGL, Thunderbird) are widely used, parsing and preprocessing vary greatly: many works even rely on "official" template sets unattainable in real deployments, yielding overly optimistic results [8], [25]. Second, the Loglizer toolkit [26], though useful as a repository of baseline models, omits preprocessing pipelines, forcing researchers to reimplement cleaning and parsing. Third, Loghub [27] centralizes popular datasets but documents HDFS preprocessing poorly, as noted by Landauer et al. [8]. These gaps hinder reproducible, end-to-end benchmarking across parsing, representation, and model choices.

III. Evaluation Pipeline

We propose a modular, end-to-end pipeline that ingests raw logs and produces comparable anomaly-detection metrics through five stages: (1) parsing to extract templates; (2) sequence grouping to aggregate templates via configurable strategies; (3) representation to transform sequences into feature vectors; (4) detector integration, which wraps detectors in a unified interface with Optuna-tuned hyperparameters [10]; and (5) unified evaluation applying consistent benchmarks. This design allows each part to be developed and extended independently while preserving fair performance comparisons.

Raw logs are preprocessed with regular expressions and parsed using Drain [9], in line with prior work. As parsing accuracy strongly impacts anomaly-detection performance [9], [28], the pipeline is structured to accommodate other parsers if needed.

Each parsed line is grouped into a sequence using either time windows or sequence IDs: for HDFS we use block IDs, while for BGL and Thunderbird we apply a sliding window of one minute or 40 lines (see Subsection IV-A2), balancing context and granularity. This design ensures that all methods operate on identical sequence definitions.

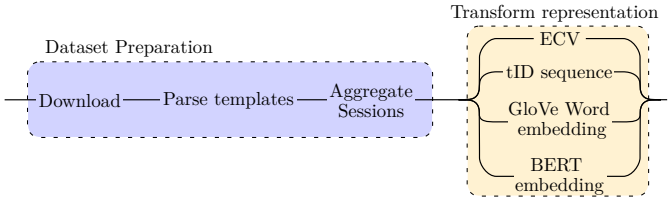

Fig. 1. **Pipeline architecture: Data preprocessing**

Each template sequence is mapped to one of four feature representations for fair comparison across detectors. (1) Event-count vectors (ECVs) aggregate template occurrences into fixed-length counts. (2) Template ID sequences preserve log

order for sequence models. (3) Summed GloVe embeddings yield dense representations capturing semantic similarity. (4) Transformer-based contextual embeddings (e.g., BERT) encode full-line context with numeric tokens removed. By holding representation type as the only variable, we directly assess the impact of feature choice on detection (see Figure 1).

We integrate classical detectors (PCA, SVM) and sequence models (LSTM predictors, transformer-based classifiers) under a common Adapter interface. Each Adapter implements `fit()` for training, `predict()` for scoring, and `preprocess_split()` for model-specific transformations. Hyperparameters are tuned on a held-out validation split using a two-phase Optuna search [10]: first adjusting training-related settings to minimize validation loss, then fine-tuning method-specific parameters (e.g., anomaly thresholds) to maximize validation F_1. This standardizes optimization across methods while preventing test-data leakage. Final models are then evaluated with 10-fold cross-validation, applying preprocessing in each fold (Figure 2).

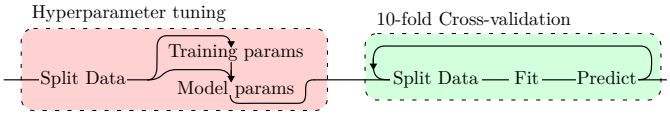

Fig. 2. **Pipeline architecture: Hyperparameter tuning and evaluation**

In our experiments, we always hold out a 10% split for validation and the remainder after the training split for testing. We note precision, recall (true positive rate), true negative rate, and binary F_1 (anomalies as the positive class), reporting F_1 as the primary metric given class imbalance. By fixing data splits, sequence definitions, and metrics across experiments, we provide a reproducible and fair benchmark of log-anomaly detectors.

IV. Experiments and Evaluation

We evaluate all anomaly-detection methods using the modular pipeline described in Section III. We first proceed to answer RQ1, comparing log anomaly detection methods under a unified evaluation protocol. We apply 10-fold shuffled cross-validation with a 50% training ratio on each dataset. We report the median F_1 along with the interquartile range (IQR) between folds. By fixing the sampling strategy (shuffled splits at a 50% training ratio) and using identical sequence definitions and hyperparameter tuning procedures, we ensure a fair comparison.

We conduct experiments on three standard log-anomaly datasets: HDFS, BGL, and Thunderbird, summarized in Table I. HDFS and BGL are widely used in prior evaluations (e.g., [1], [3], [4], [29]), while Thunderbird appears less frequently. HDFS contains about 11.2M lines grouped into 575,061 sequences with a single anomalous label and 46–50 templates (depending on the source) [17], [27]. BGL has about 4.7M lines and 134,975 sequences with 43 anomalous labels across 298 templates [30]. For Thunderbird, we use the first 20M lines due to computational limits, forming 500,965 sequences with 21 anomalous labels and 1,874 templates.

TABLE I
Used datasets. *Labels* shows the count of anomaly labels. We describe the only used sample of the Thunderbird dataset.

Dataset Original paper	**Log lines**	**Sequences** Anomalous	**Labels**	**Templates**
HDFS (Xu)	11,197,705	575,061	1	50 (148)
Xu et al. [17]		16,838		
HDFS (LogHub)	11,175,629	575,061	1	46
Zhu et al. [27]		16,838		
BGL	4,747,963	134,975	43	298
Oliner et al. [30]		11,116		
Thunderbird	20,000,000	500,965	21	1,874
Oliner et al. [30]		156,017		

TABLE II
Summary of selected anomaly detection methods

Representation	**Unsupervised**	**Semi-supervised**	**Supervised**
ECV	PCA	LogCluster [21]	SVM [23]
tID sequence	–	DeepLog [6]	–
Word emb.	SemPCA [25]	LogAnomaly [4]	LogRobust [5]
Contextual emb.	–	LogBERT [7]	NeuralLog [16]

We evaluate nine anomaly-detection models (Table II) spanning unsupervised, semi-supervised, and supervised approaches with different data representations. Table III reports their results on all three datasets.

Supervised models achieve the highest median F_1, though the leader varies by dataset: LogRobust on HDFS, NeuralLog on BGL, and near-identical performance across models on Thunderbird, where SVM and LogRobust tie. Among semi-supervised methods, LogBERT yields moderate scores but falls behind supervised baselines on HDFS and BGL. LogAnomaly and DeepLog show a similar pattern, stronger on Thunderbird and weaker on HDFS and BGL. LogCluster performs best among semi-supervised models on HDFS but lags on the others. Unsupervised PCA-based methods diverge sharply: SemPCA is competitive on HDFS but drops on BGL and Thunderbird, while PCA performs worst overall.

When comparing to related work, our evaluation of NeuralLog closely reproduces the results of Le et al. [16]. LogRobust

TABLE III
Median F_1 score (IQR) for each anomaly-detection method under 10-fold shuffled cross-validation on HDFS, BGL, and Thunderbird. Best results per dataset in bold.

Method	**HDFS**	**BGL**	**TBird**
NeuralLog	0.989 (0.005)	**0.980 (0.017)**	0.998 (0.000)
LogRobust	**0.996 (0.002)**	0.940 (0.022)	**0.999 (0.000)**
SVM	0.977 (0.002)	0.917 (0.003)	**0.999 (0.000)**
LogBERT	0.733 (0.013)	0.790 (0.009)	0.968 (0.001)
LogAnomaly	0.912 (0.024)	0.774 (0.016)	0.948 (0.001)
DeepLog	0.890 (0.085)	0.769 (0.011)	0.947 (0.001)
LogCluster	0.931 (0.004)	0.761 (0.003)	0.448 (0.147)
SemPCA	0.944 (0.015)	0.448 (0.006)	0.485 (0.001)
PCA	0.810 (0.025)	0.440 (0.003)	0.379 (0.002)

matches the findings of Zhang et al. [5] and others [16], [25].

LogAnomaly performs near the original results of Meng et al. [4], exceeding some later reports [7], [25]. DeepLog does not replicate the numbers of Du et al. [6] but aligns with related work [7], [25]. SemPCA reproduces Yang et al.'s HDFS results [25] but differs on BGL due to preprocessing. Our LogCluster results also match Yang et al. [25] and outperform Meng et al. [4] and Guo et al. [7].

Despite available source code, we do not reproduce LogBERT's reported results, trailing Guo et al. [7] by about 10%.

Finally, we confirm SVM as a strong supervised baseline, though rarely emphasized in prior work, while PCA is a competitive unsupervised benchmark on HDFS but weaker on the other datasets.

All hyperparameters for the tested methods are published in our released source code. We note one exception in hyperparameter tuning: for next-event prediction methods (DeepLog, LogAnomaly, LogBERT), the number of candidates (*top-k*) was originally tuned per dataset. This sometimes produced invalid settings when k exceeded the number of unique classes. To give each method the best chance, we instead treat k as a learned parameter, tuning it on the training split of each fold under full supervision. While this slightly favors these semi-supervised methods, it ensures reported F_1 scores reflect their best achievable performance.

A. Influence of preprocessing

This section investigates RQ2: How do choices in preprocessing, such as dataset filtering or sequence grouping, affect anomaly-detection performance? We study two cases. In HDFS, merging or removing problematic templates reduces template explosion and can inflate performance by making anomalies easier to isolate. In BGL, grouping by component improves detection but risks long latency, while combining line-count and time-window constraints shortens sequences for timely detection at the cost of lower F_1 scores.

1) HDFS dataset preprocessing: The original HDFS logs (*Xu*) contain a complex template with a dynamic number of parameters, which Drain cannot parse reliably. These logs record requests to delete variable-length block lists (1–100 blocks). Because Drain partitions lines by token length, it generates a new template for each sequence length, producing 98 templates for this event and inflating the total from 50 to 148. This template explosion increases feature dimensionality and randomness, severely degrading next-event prediction and masked-event models. In the *Fixed* version, we retain these events but remap all scattered templates to the shortest one, correcting the parser. As Table IV shows, next-event methods (LogBERT, LogAnomaly, DeepLog) lose at least 20 F_1 points on the *Xu* logs, while PCA drops by more than 40 points compared to the *Fixed* dataset.

LogHub provides an another HDFS variant that removes the problematic "block-delete" events, nine "DataXceiver" lines, and 34 "No such file or directory" lines, shortening the dataset by about 22,000 lines. This filtering artificially boosts anomaly-detection performance, as also noted by Landauer et al. [8], since many removed events appear in both anomalous and normal sequences, making separation without them easier. As Table IV shows, most methods score higher on *LogHub* than on our *Fixed* variant (which only merges dynamic templates): PCA rises from 0.72 to 0.81, DeepLog from 0.83 to 0.89, while LogRobust stays near perfect (0.99–1.00). LogCluster is the exception, dropping slightly by 0.6%.

TABLE IV
MEDIAN F_1 SCORE (IQR) FOR EACH ANOMALY-DETECTION METHOD UNDER 10-FOLD SHUFFLED CROSS-VALIDATION ON HDFS WITH VARIOUS PREPROCESSING OPTIONS. BEST RESULTS PER DATASET IN BOLD.

Method	HDFS Xu	HDFS Fixed	HDFS LogHub
NeuralLog	0.971 (0.011)	0.971 (0.014)	0.989 (0.005)
LogRobust	**0.994 (0.003)**	**0.991 (0.003)**	**0.996 (0.002)**
SVM	0.977 (0.001)	0.986 (0.005)	0.977 (0.002)
LogBERT	0.494 (0.031)	0.710 (0.030)	0.733 (0.013)
LogAnomaly	0.698 (0.009)	0.892 (0.028)	0.912 (0.024)
DeepLog	0.607 (0.043)	0.828 (0.035)	0.890 (0.085)
LogCluster	0.799 (0.003)	0.937 (0.005)	0.931 (0.004)
SemPCA	0.909 (0.062)	0.937 (0.003)	0.944 (0.015)
PCA	0.268 (0.004)	0.724 (0.025)	0.810 (0.025)

For comparability with prior work, all remaining experiments use the *LogHub* version. This choice yields higher F_1 than raw logs but ensures alignment with existing studies.

2) BGL grouping strategies: There is no consensus on how to group BGL log lines into sequences for anomaly detection. Some studies adopt component-based grouping [8], [25], [31], while others use time-based windows or hybrids combining time and line limits [32]. For example, NeuralLog applies sliding windows of 20 lines without component boundaries [16]; Guo et al. use a 5-minute time window followed by 128-line splits [7]; and Le et al. compare fixed windows of 20, 100, and 200 lines, concluding that component grouping performs best [31]. Other works omit their grouping strategy altogether [1], [4], limiting reproducibility.

Fixed-line windows ensure each sequence has exactly L log entries, but their temporal span varies widely: a 40-line window may cover hours during low activity or milliseconds during bursts. Pure time windows show the opposite imbalance: quiet periods yield many single-line sequences, while busy periods produce extremely large ones. In our tests with a one-hour window, we observed 150 one-line sequences and 141 sequences with 5,000–100,000 lines; shortening the interval increased the one-line cases to thousands without removing the long tail. These results confirm that BGL's event rate is highly variable, creating sequences that either lack context or overwhelm next-event models. Consequently, pure time windows are impractical for anomaly detection.

Combining a line-count limit with a time limit mitigates both extremes by capping sequences at L lines or T seconds. However, our tests show this windowing is incompatible with component grouping: applying a 120-line/5-minute window with component grouping produced 1,310,188 one-line sequences (27% of lines, 54% of sequences).

Table V lists the evaluated window settings, alongside the 120-line fixed window with component grouping used by Yang

TABLE V
SEQUENCE LENGTHS ON BGL UNDER VARIOUS GROUPING OPTIONS.

Grouping	Lines ($\mu \pm \sigma$)	Time span ($\mu \pm \sigma$)
Component & 120l	66.09 ± 43.74	27 d:19.5 h $\pm$ 7 d:14.2 h
120l & 60s	84.14 ± 50.88	11.4 s $\pm$ 17.7 s
40l & 60s	35.89 ± 11.53	4.8 s $\pm$ 12.1 s

TABLE VI
SUMMARY OF ANOMALY COUNTS WITH EACH GROUPING STRATEGY. THE LAST COLUMN SHOWS THE COUNT OF ANOMALOUS LINES PER SEQUENCE FOR ANOMALOUS SEQUENCES ONLY.

Grouping	Sequences			A. per Sequence
	Total	Normal	Anomalous	($\mu \pm \sigma$)
Comp. & 120l	85 577	49 094	36 483	9.56 ± 22.41
120l & 60s	58 124	53 533	4 591	75.95 ± 52.33
40l & 60s	134 975	123 859	11 116	31.37 ± 14.43

et al. [25]. The mixed windows yield sequences short enough for near–real-time detection while retaining sufficient context for anomaly detectors to learn patterns.

Beyond length and duration, grouping strategy also determines how many anomalous lines fall into each anomalous sequence. Table VI shows the total, normal, and anomalous sequence counts per strategy. The trend is clear: as window granularity increases, the share of anomalous sequences falls (42.6%, 7.9%, 8.2%), while the mean fraction of anomalous lines per anomalous sequence rises (12.7%, 75.9%, 83.1%).

Table VII reports results for component grouping, a 120-line/60-s window, and a 40-line/60-s window. As expected, component grouping is the easiest: all methods achieve their highest F_1. Yet its long spans make it impractical for timely detection, since anomalies are visible only after sequence completion, which may take hours. Between 120- and 40-line windows, outcomes vary by method. LogRobust, SVM, and PCA are largely unaffected. LogAnomaly and LogCluster favor 120 lines, while DeepLog and SemPCA perform better with 40; SemPCA improves by nearly 20% and shows lower variance. LogBERT loses about four points in the 40-line case (precision stable, recall down 7–8%), reflecting sensitivity to shorter sequences and reliance on masking. NeuralLog gains most from 40 lines, improving by $\sim$10% mainly via fewer false negatives, with slight precision gains. In sum, while component grouping maximizes F_1, mixed windows balance realistic latency with performance.

Based on these findings, we adopt the 40-line/60-second fixed window for all other BGL experiments. This keeps sequences short enough for timely detection while preserving sufficient context for classification. We acknowledge it disadvantages LogBERT, which struggles with short sequences [7], though LAnoBERT mitigates this by predicting over all tokens [1]. Shorter windows (e.g., 20 lines as in NeuralLog's evaluation) are also reasonable for models sensitive to sequence length; we leave detailed exploration to future work.

TABLE VII
MEDIAN F_1 SCORE (IQR) FOR EACH METHOD UNDER 10-FOLD SHUFFLED CROSS-VALIDATION ON BGL WITH VARIOUS GROUPING OPTIONS.

Method	Component & 120l	120l & 60s	40l & 60s
NeuralLog	0.986 (0.003)	0.863 (0.012)	**0.980 (0.017)**
LogRobust	0.998 (0.003)	**0.930 (0.017)**	0.940 (0.022)
SVM	**1.000 (0.000)**	0.924 (0.006)	0.917 (0.003)
LogBERT	0.983 (0.001)	0.834 (0.010)	0.790 (0.009)
LogAnomaly	0.827 (0.024)	0.781 (0.020)	0.774 (0.016)
DeepLog	0.834 (0.037)	0.729 (0.047)	0.769 (0.011)
LogCluster	0.946 (0.002)	0.789 (0.005)	0.761 (0.003)
SemPCA	0.674 (0.001)	0.294 (0.141)	0.448 (0.006)
PCA	0.599 (0.004)	0.433 (0.004)	0.440 (0.003)

B. Influence of Data-splitting Procedures

In this section, we address RQ3: how does training on a randomly shuffled sample compare to training on a temporally continuous block of logs, and how do these differences change as the training set size decreases?

We use 10-fold cross-validation that preserves temporal continuity when required (Figure 3). Each fold consists of three continuous dataset segments: 50%, 10%, or 1% for training, 10% for validation, and the remainder for testing. We evaluate two sampling methods: in the *shuffled* setting, sequences are randomly shuffled before splitting, as in prior experiments and much of the literature; in the *sequential* setting, sequence order is preserved so train and test data come from disjoint time intervals, reflecting real-world deployment.

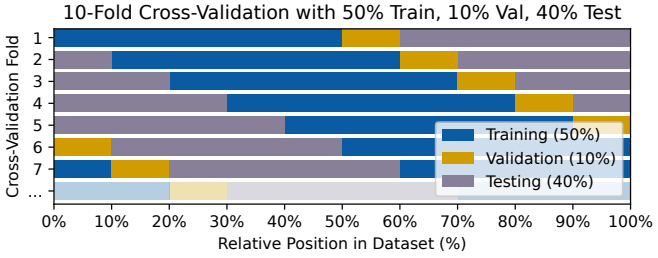

Fig. 3. **Folding of datasets in 10-fold cross-validation.** Folds 1-7 are shown; folds 8-10 are omitted for brevity.

At a 50% training ratio (Table VIII), all models perform better with shuffled than sequential sampling, though the gap depends on the dataset. On HDFS, supervised detectors such as LogRobust and SVM lose little under sequential sampling, while NeuralLog drops by about 7%. On Thunderbird, supervised methods also degrade only slightly, whereas others fall more sharply; PCA is the exception, improving modestly but still performing poorly. Overall, the drop is larger on HDFS than Thunderbird, but BGL is hit hardest: under sequential sampling, all models fall below acceptable levels. This is due to concept drift, shifts in normal system behavior during data capture, which detectors misclassify as anomalies, driving false positives, while unseen anomaly types cause supervised methods to miss anomalies. The impact varies across folds, reflected in high IQR values.

TABLE VIII
MEDIAN F_1 SCORE (IQR) FOR EACH ANOMALY-DETECTION METHOD ON HDFS AT 50% AND 1% TRAINING RATIOS UNDER SHUFFLED AND SEQUENTIAL SPLITS, ALONGSIDE 50%-ONLY RESULTS FOR BGL AND THUNDERBIRD. BEST VALUES IN EACH COLUMN ARE **BOLDED**.

Method	HDFS				BGL (50%)		TBird (50%)	
	Shuf 50%	Seq 50%	Shuf 1%	Seq 1%	Shuf	Seq	Shuf	Seq
NeuralLog	0.989 (0.005)	0.917 (0.112)	0.000 (0.000)	0.000 (0.380)	**0.980** (0.017)	0.490 (0.690)	0.998 (0.000)	0.993 (0.006)
LogRobust	**0.996** (0.002)	**0.954** (0.061)	**0.958** (0.018)	0.517 (0.295)	0.940 (0.022)	0.189 (0.323)	**0.999** (0.000)	0.993 (0.007)
SVM	0.977 (0.002)	0.931 (0.071)	0.944 (0.033)	**0.788** (0.172)	0.917 (0.003)	**0.570** (0.111)	**0.999** (0.000)	**0.997** (0.005)
LogBERT	0.733 (0.013)	0.610 (0.417)	0.603 (0.070)	0.135 (0.241)	0.790 (0.009)	0.257 (0.328)	0.968 (0.001)	0.860 (0.139)
LogAnomaly	0.912 (0.024)	0.706 (0.475)	0.933 (0.009)	0.190 (0.568)	0.774 (0.016)	0.380 (0.290)	0.948 (0.001)	0.777 (0.158)
DeepLog	0.890 (0.085)	0.577 (0.558)	0.775 (0.086)	0.421 (0.689)	0.769 (0.011)	0.350 (0.344)	0.947 (0.001)	0.808 (0.120)
LogCluster	0.931 (0.004)	0.902 (0.602)	0.931 (0.010)	0.141 (0.056)	0.761 (0.003)	0.420 (0.305)	0.448 (0.147)	0.472 (0.191)
SemPCA	0.944 (0.015)	0.693 (0.090)	0.943 (0.001)	0.099 (0.033)	0.448 (0.006)	0.176 (0.061)	0.485 (0.001)	0.214 (0.278)
PCA	0.810 (0.025)	0.795 (0.180)	0.790 (0.007)	0.147 (0.356)	0.440 (0.003)	0.279 (0.184)	0.379 (0.002)	0.522 (0.235)

TABLE IX
MEDIAN F_1 SCORE (IQR) AT 10% TRAINING RATIO FOR HDFS AND BGL UNDER SHUFFLED AND SEQUENTIAL SPLITS.

Method	HDFS		BGL (shuffled)	
	Shuf 10%	Seq 10%	10%	1%
NeuralLog	0.91 (0.02)	0.27 (0.59)	0.89 (0.02)	0.00 (0.00)
LogRobust	**0.99** (0.00)	0.83 (0.18)	0.90 (0.01)	0.84 (0.03)
SVM	**0.99** (0.01)	**0.91** (0.03)	**0.91** (0.00)	**0.88** (0.02)
LogBERT	0.75 (0.03)	0.20 (0.35)	0.81 (0.02)	0.76 (0.01)
LogAnomaly	0.95 (0.01)	0.46 (0.75)	0.79 (0.01)	0.75 (0.01)
DeepLog	0.82 (0.05)	0.24 (0.52)	0.79 (0.01)	0.75 (0.01)
LogCluster	0.93 (0.00)	0.21 (0.62)	0.78 (0.00)	0.71 (0.01)
SemPCA	0.96 (0.02)	0.18 (0.04)	0.38 (0.10)	0.31 (0.16)
PCA	0.79 (0.00)	0.25 (0.53)	0.44 (0.00)	0.44 (0.01)

This is underscored by the HDFS results at 1% training ratio (Table VIII). Under sequential sampling, most methods degrade sharply compared to both 50% sequential and 1% shuffled benchmarks. SVM remains relatively robust with a median F_1 of 0.788 (vs. 0.944 at 1% shuffled), while LogRobust drops by 43% and NeuralLog collapses to zero in both 1% settings. Inspection shows NeuralLog predicts only the normal class, indicating class imbalance rather than dataset size is the issue. In contrast, semi- and unsupervised methods perform on par with 50% shuffled samples, with LogAnomaly even improving its median F_1 by 2.1% at 1%. Overall, a 1% random sample yields better median performance than a 50% continuous block for most methods on HDFS.

Additional results are shown in Table IX. On shuffled BGL, median F_1 decreases steadily as the training ratio shrinks, with SVM as the exception: even at 1% it loses only 3.6% relative to 50%, reaching the best score of 88%. NeuralLog again collapses to 0.0 at 1%. We omit sequential BGL at lower ratios since performance at 50% is already below an acceptable threshold. HDFS follows the same trends as in Table VIII.

These results show that models trained on shuffled logs consistently outperform those trained on continuous blocks, even with less data. On HDFS and BGL, 1% shuffled often exceeds 50% sequential (e.g., LogRobust, DeepLog, SemPCA). This suggests that the diversity captured by shuffling outweighs the benefit of longer windows. In practice, shuffling periodically sampled logs for training should yield more reliable detection than retraining only on recent continuous segments.

V. CONCLUSION

We present a modular, end-to-end log-anomaly evaluation pipeline that unifies parsing, sequence grouping, representation, and detector integration in an open-source framework. Our benchmarks show that transformer-based models excel on shuffled data but degrade on time-ordered logs, while classical methods such as SVM remain stable. We find that minor preprocessing choices, particularly sequence grouping, substantially shift performance. These results highlight the need for standardized evaluation: only under identical preprocessing and splitting can gains be attributed to true improvements rather than evaluation artifacts. Future works could improve target system-specific dense-vector representations and explore if optimal sequence windows depend on method or dataset.

ACKNOWLEDGMENT

This work was supported by the Technology Agency of the Czech Republic under grant agreement No FW12010526.

REFERENCES

[1] Y. Lee, J. Kim, and P. Kang, "LAnoBERT: System log anomaly detection based on BERT masked language model," *Applied Soft Computing*, vol. 146, Oct. 2023. DOI: 10.1016/j.asoc.2023.110689.

[2] S. Chen and H. Liao, "BERT-Log: Anomaly Detection for System Logs Based on Pre-trained Language Model," *Applied Artificial Intelligence*, vol. 36, no. 1, Dec. 2022. DOI: 10.1080/08839514.2022.2145642.

[3] S. Huang, Y. Liu, C. Fung, *et al.*, "HitAnomaly: Hierarchical Transformers for Anomaly Detection in System Log," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, Dec. 2020. DOI: 10.1109/TNSM.2020.3034647.

[4] W. Meng, Y. Liu, Y. Zhu, *et al.*, "LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, Jul. 2019. DOI: 10.24963/ijcai.2019/658.

[5] X. Zhang, Y. Xu, Q. Lin, *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*, ser. ESEC/FSE 2019, ACM, Aug. 2019. DOI: 10.1145/3338906.3338931.

[6] 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*, ser. CCS '17, ACM, Oct. 2017. DOI: 10.1145/3133956.3134015.

[7] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log Anomaly Detection via BERT," in *2021 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2021. DOI: 10.1109/IJCNN52387.2021.9534113.

[8] M. Landauer, F. Skopik, and M. Wurzenberger, "A Critical Review of Common Log Data Sets Used for Evaluation of Sequence-Based Anomaly Detection Techniques," *Reproduction package for article "A Critical Review of Common Log Data Sets Used for Evaluation of Sequence-Based Anomaly Detection Techniques"*, vol. 1, no. FSE, Jul. 2024. DOI: 10.1145/3660768.

[9] 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)*, Jun. 2017. DOI: 10.1109/ICWS.2017.13.

[10] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[11] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No.03EX764)*, Oct. 2003. DOI: 10.1109/IPOM.2003.1251233.

[12] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09, ACM, Jun. 2009. DOI: 10.1145/1557019.1557154.

[13] M. Du and F. Li, "Spell: Streaming Parsing of System Event Logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Dec. 2016. DOI: 10.1109/ICDM.2016.0103.

[14] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-supervised Log Parsing," in *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track*, Y. Dong, D. Mladenić, and C. Saunders, Eds., Cham: Springer Int. Pub., 2021. DOI: 10.1007/978-3-030-67667-4_8.

[15] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "SwissLog: Robust and Unified Deep Learning Based Log Anomaly Detection for Diverse Faults," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, Oct. 2020. DOI: 10.1109/ISSRE5003.2020.00018.

[16] 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)*, Nov. 2021. DOI: 10.1109/ASE51524.2021.9678773.

[17] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, ser. SOSP '09, ACM, Oct. 2009. DOI: 10.1145/1629575.1629587.

[18] J. E. Prewett, "Analyzing cluster log files using logsurfer," in *Proceedings of the 4th Annual Conference on Linux Clusters*, 2003.

[19] J. P. Rouillard, "Real-time Log File Analysis Using the Simple Event Correlator (SEC).," in *LISA*, vol. 4, 2004, pp. 133–150.

[20] J.-G. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.

[21] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16, ACM, May 2016. DOI: 10.1145/2889160.2889232.

[22] M. Catillo, A. Pecchia, and U. Villano, "AutoLog: Anomaly detection by deep autoencoding of system logs," *Expert Systems with Applications*, vol. 191, Apr. 2022. DOI: 10.1016/j.eswa.2021.116263.

[23] Y. Liang, Y. Zhang, H. Xiong, and R. Sahoo, "Failure Prediction in IBM BlueGene/L Event Logs," in *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, Oct. 2007. DOI: 10.1109/ICDM.2007.46.

[24] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *International Conference on Autonomic Computing, 2004. Proceedings.*, May 2004. DOI: 10.1109/ICAC.2004.1301345.

[25] L. Yang, J. Chen, S. Gao, *et al.*, "Try with Simpler – An Evaluation of Improved Principal Component Analysis in Log-based Anomaly Detection," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 5, Jun. 2024. DOI: 10.1145/3644386.

[26] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience Report: System Log Analysis for Anomaly Detection," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, Oct. 2016. DOI: 10.1109/ISSRE.2016.21.

[27] 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*, Sep. 2023. DOI: 10.48550/arXiv.2008.06448. arXiv: 2008.06448 [cs].

[28] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-Attentive Classification-Based Anomaly Detection in Unstructured Logs," 2020.

[29] L. Yang, J. Chen, Z. Wang, *et al.*, "Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, May 2021. DOI: 10.1109/ICSE43902.2021.00130.

[30] 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)*, Jun. 2007. DOI: 10.1109/DSN.2007.103.

[31] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" In *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22, ACM, Jul. 2022. DOI: 10.1145/3510003.3510155.

[32] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, *Experience Report: Deep Learning-based System Log Analysis for Anomaly Detection*, Jan. 2022. DOI: 10.48550/arXiv.2107.05908.