PULSELOG: CONTEXT-CONDITIONED BEHAVIORAL DYNAMICS FOR NOISE-RESILIENT LOG ANOMALY DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Log anomaly detection requires inferring system behavior from observations. Their semantic and contextual evidence can be distorted by template variation, parameter errors, formatting changes, and other observation noise. Although existing methods have advanced semantic representation learning and temporal dependency modeling, they largely treat observed events as homogeneous evidence. Persistent operational regimes, transient deviations, and runtime context are entangled in a single latent trajectory. This prevents the detector from separating observation-induced perturbations from behaviorally meaningful changes. It also makes anomaly decisions sensitive to benign fluctuations and contextual shifts. We propose PulseLog, a context-conditioned behavioral dynamics model for noise-resilient log anomaly detection. PulseLog maps log observations to semantic anchors. It calibrates runtime context. It adaptively routes persistent and transient evidence at multiple temporal rates. It evolves a latent behavioral state through context-gated dynamics. A deviation probe identifies departures from context-specific state-transition patterns. This enables window-level anomaly detection based on deviations in system-state dynamics. Experiments across five public log systems demonstrate strong detection performance under standard evaluation, observation-noise perturbations, limited training data, and cross-system adaptation. PulseLog also has a compact model footprint. The results indicate that explicitly modeling context-conditioned behavioral dynamics improves robustness under the evaluated log observation conditions.

1 INTRODUCTION

Large-scale software systems continuously emit logs that record executions, interactions among components, and failures. These records are often the most accessible evidence for diagnosing abnormal system behavior. This makes log anomaly detection a central capability in modern operations (Du et al., 2017; Meng et al., 2019; Le & Zhang, 2024; Zhou et al., 2022; Zhang et al., 2023; Xie et al., 2022; Tao et al., 2026). Yet the target of detection is not an isolated unusual message. An event that appears uncommon can be fully consistent with an ongoing workload. A familiar event can instead signal a failure when it occurs under an incompatible runtime condition. Effective detection therefore requires modeling how system behavior evolves over time. It also requires determining whether a current observation is consistent with that behavior.

This task is complicated by the fact that the evidence available in logs is noisy. Template variation, parameter corruption or truncation, unseen tokens, format changes, and occasional record perturbations can alter the local semantic and contextual cues used to infer behavior (Zhang et al., 2019; Le & Zhang, 2021; Khan et al., 2024; Zhu et al., 2019; Zhou et al., 2022; Xie et al., 2022; Zhang et al., 2023). The problem arises across representation choices. Template-based pipelines can propagate local inaccuracies from preprocessing. Methods operating directly on log text still encounter evolving language, formatting changes, and noisy runtime arguments (Zhang et al., 2019; Le & Zhang, 2021). At the same time, normal execution is not temporally uniform. It contains persistent operational regimes, short-lived transitions, and bursts. A useful detector must suppress locally unreliable evidence without erasing transient deviations that reveal an anomaly.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Prior work has substantially advanced log anomaly detection through sequential prediction, semantic representations, reconstruction objectives, graph-based dependency modeling, and self-supervised representation learning (Du et al., 2017; Meng et al., 2019; Le & Zhang, 2021; Chu et al., 2025; Xie et al., 2024; Guo et al., 2021; Xu et al., 2021; Tuli et al., 2022; Yue et al., 2022; Yang et al., 2023; Wu et al., 2022; Nie et al., 2022; Liu et al., 2024a). Methods have also explored semi-supervised learning, weak supervision, and transfer-oriented pre-training to reduce dependence on abundant target-domain labels (Yang et al., 2021; He et al., 2025; Guo et al., 2024; Zhou et al., 2025; Zhang et al., 2024; Landauer et al., 2023; Zhou et al., 2023; Das et al., 2023; Wang et al., 2024; Luo & Wang, 2024; Goswami et al., 2024; Liu et al., 2024b; Jin et al., 2024). However, these directions leave a common modeling gap. Existing detectors typically encode observations within one latent temporal trajectory. Persistent operational evidence, transient changes, and runtime context jointly compete for the same representation. This entanglement makes it difficult to identify whether a local shift reflects observation noise, benign short-term variation, or a context-dependent behavioral deviation. Moreover, runtime parameters are often appended as auxiliary features. They are not used to condition the evolution of the latent behavioral state. Consequently, the learned state-transition representation remains insufficiently tied to the runtime conditions under which the event occurs.

We formulate noise-resilient log anomaly detection as context-conditioned behavioral state dynamics. Each window contains noisy semantic and runtime-context evidence. The objective is to model how system states evolve under context and detect deviations from learned context-specific transition patterns. This perspective treats individual log observations as imperfect measurements. Anomalies are deviations in the state transitions they collectively indicate. It motivates a temporal decomposition. Persistent evidence stabilizes state evolution. Transient evidence captures abrupt but meaningful changes. Runtime context modulates state transitions. Based on this formulation, we propose PulseLog. It maps logs to Semantic Anchors, derives runtime conditions through a Context Calibrator, and organizes persistent and transient evidence with a Multi-Rate Evidence Router. Context-Gated Dynamics models latent state evolution under runtime context. A Deviation Probe identifies behavioral inconsistency from state changes. Thus, anomaly decisions are based on learned behavioral dynamics rather than isolated event features. Experiments show that PulseLog achieves strong detection performance and robustness under the evaluated log observation conditions.

Our contributions are threefold:

1. We formulate noise-resilient log anomaly detection as context-conditioned behavioral state dynamics, in which anomaly decisions are determined by deviations in system-state transitions from context-specific patterns learned from data.
2. We propose PulseLog, which combines semantic anchors, runtime-context calibration, multi-rate evidence routing, and context-gated state dynamics to model persistent and transient behavioral evidence under observation noise.
3. We provide a stress-test evaluation beyond standard in-domain detection, covering observation noise, reduced training data, cross-system adaptation, computational cost, representation analysis, and component attribution.

2 Related Work

2.1 Log Anomaly Detection

Existing log anomaly detectors mainly learn regularities in event sequences, message semantics, or system structure. DeepLog predicts the next log event from preceding events, while LogAnomaly jointly models sequential and quantitative patterns (Du et al., 2017; Meng et al., 2019). Parser-free NeuralLog uses the semantics of raw log messages, and LogBERT and LogSD learn contextual representations through masked or self-supervised objectives (Le & Zhang, 2021; Guo et al., 2021; Xie et al., 2024). Lograph further incorporates associations between log messages and system entities through a graph representation (Chu et al., 2025). These methods provide complementary ways to represent log evidence, but they generally combine persistent operational patterns and transient changes in a shared temporal representation. PulseLog explicitly separates persistent and transient evidence across multiple temporal rates and incorporates runtime context into latent state evolution, enabling window-level decisions to reflect changes in system behavior rather than undifferentiated event representations.

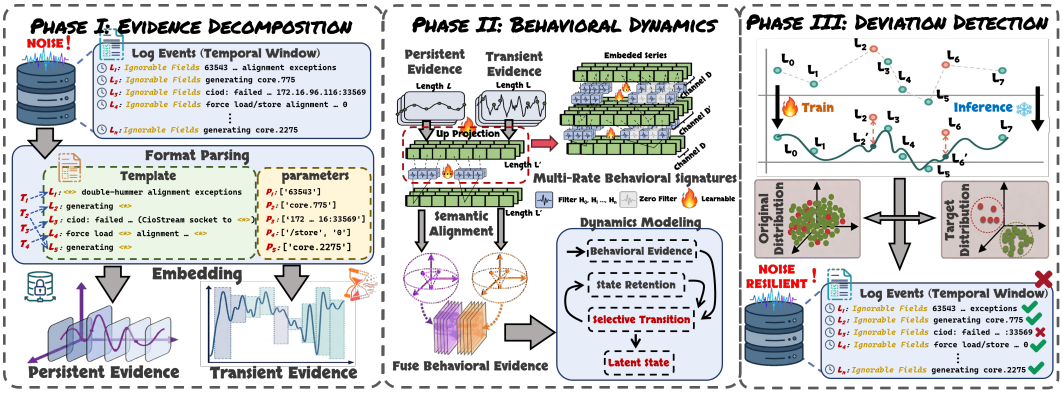

Figure 1: Overview of PulseLog. Noisy log observations are parsed into template semantics and runtime parameters, encoded as event representations, and organized into Persistent evidence and transient evidence through Multi-Rate Behavioral Signatures. Runtime context conditions the latent state transitions, whose complete trajectory is read by a Deviation Probe to produce a window-level behavioral transition deviation.

2.2 Robustness to Noise and Limited Labels

Real-world log observations are affected by evolving templates, parameter errors, formatting changes, and parser-induced perturbations. RobustLog studies anomaly detection on unstable log data, while NeuralLog avoids dependence on a fixed parser and analyzes the effect of parsing errors on detection (Zhang et al., 2019; Le & Zhang, 2021). Parsing benchmarks and empirical studies further show that parsing outcomes vary across methods and that parsing quality does not directly determine downstream anomaly detection quality (Zhu et al., 2019; Khan et al., 2024). In parallel, PLELog and MIDLog address the scarcity or imprecision of anomaly labels through probabilistic label estimation and multi-instance learning (Yang et al., 2021; He et al., 2025); PreLog and LogFormer reduce target-task supervision through log pre-training (Le & Zhang, 2024; Guo et al., 2024). These methods address observation noise and label scarcity through parser robustness, alternative supervision, or representation transfer, but they do not explicitly model how noisy observations and limited training data affect the evolution of system behavior. PulseLog instead learns the evolution of a latent behavioral state from Persistent evidence and transient evidence, while using runtime context to modulate this evolution. This design supports anomaly decisions based on learned behavioral inconsistency under the evaluated noisy and data-limited settings.

3 Methodology

3.1 Problem Formulation

Let x_t denote the raw log message at time t. For a window of T events, the available observation is $\widetilde{\mathcal{O}}_{1:T} = \{\widetilde{o}_t\}_{t=1}^{T}$ with $\widetilde{o}_t = (\widetilde{s}_t, \widetilde{c}_t)$, where $\widetilde{s}_t$ and $\widetilde{c}_t$ are noisy observation components containing template semantics and runtime parameters. The tildes indicate that the observation can be distorted by template variation, parameter corruption or truncation, unknown terms, and formatting changes. Each window has a binary label $y \in \{0, 1\}$.

Let $h_t \in \mathbb{R}^d$ be the latent behavioral state, ξ_t the fused evidence, and r_t the runtime context representation. PulseLog defines a deterministic runtime-context-conditioned latent state transition

$$p_\Theta(h_t \mid h_{t-1}, \xi_t, r_t) = \delta\big(h_t - \mathcal{F}_\Theta\big(h_{t-1}, \xi_t; \Gamma_\Theta(r_t)\big)\big)\,, \tag{1}$$

where $\mathcal{F}_\Theta$ is the latent state transition map and $\Gamma_\Theta(r_t)$ generates runtime-context-dependent latent state transition controls. Runtime context therefore modulates latent state transitions directly: two semantically similar events can produce different state changes under different runtime parameters. The decision is based on the path $\mathcal{H}_{1:T} = (h_1, \Delta h_2, \ldots, \Delta h_T)$, where $\Delta h_t = h_t - h_{t-1}$, so the

detector evaluates the behavioral transition deviation across the window rather than relying on the standalone contribution of a rare anomalous template's semantics.

3.2 Behavioral Evidence from Log Observations

We obtain the representations used to form Persistent evidence and transient evidence with Drain (He et al., 2017b), Sentence- (Reimers & Gurevych, 2019), and a character-level encoder. Drain maps each message to a template-related component and runtime parameters, $\mathcal{P}(x_t) = (\widetilde{\tau}_t, \widetilde{a}_t)$. The Semantic Anchor is the projected semantic representation $a_t = W_a f_{\mathrm{SB}}(\widetilde{\tau}_t) + b_a \in \mathbb{R}^d$, which maps template semantics to the shared dynamics space. The Context Calibrator maps runtime parameters to a bounded control vector $r_t = \sigma(W_c f_{\mathrm{char}}(\widetilde{a}_t) + b_c) \in (0,1)^{d_c}$. Here W_a, b_a, W_c, b_c are learned jointly with the detector. Drain provides an imperfect structural prior; the character encoder retains open-vocabulary patterns such as identifiers, paths, addresses, and numbers. The pair (a_t, r_t) feeds the Multi-Rate Behavioral Signatures and context-conditioned transitions.

3.3 Multi-Rate Context-Conditioned State Dynamics

A log window captures both Persistent and transient evidence, reflecting regime-level continuity and localized changes. We normalize event time as $u_t = t/T$. For each branch $m \in \mathrm{per}, \mathrm{tr}$, we use the normalized weights $w_{m,tj}$ above, with continuous aggregation implemented as a weighted sum over the T events.

$$w_{m,tj} = \frac{\exp\!\Big(\alpha_m r_t^\top r_j - \frac{(u_t-u_j)^2}{2\lambda_m^2}\Big)}{\sum_{k=1}^{T} \exp\!\Big(\alpha_m r_t^\top r_k - \frac{(u_t-u_k)^2}{2\lambda_m^2}\Big)}, \qquad \bar{a}_{m,t} = \sum_{j=1}^{T} w_{m,tj} a_j, \tag{2}$$

where α_m is a learned context scale, and $\lambda_{\mathrm{per}} > \lambda_{\mathrm{tr}}$ controls broader persistent and localized transient support. The signatures fuse each summary with the current context through learned maps ϕ_m. The normalized weights enable full-window aggregation: the persistent branch captures broader temporal patterns, while the transient branch focuses on local changes. Renormalization over $j = 1, \ldots, T$ handles window boundaries without padding.

$$p_t = \phi_{\mathrm{per}}\!\left(\int_0^1 K_{\mathrm{per}}(u_t, v; r) e(v)\, dv,\ r(u_t)\right), \qquad q_t = \phi_{\mathrm{tr}}\!\left(\int_0^1 K_{\mathrm{tr}}(u_t, v; r) e(v)\, dv,\ r(u_t)\right). \tag{3}$$

The two signatures are jointly optimized with the detector to separate normal and anomalous trajectories: p_t captures regime-level consistency, while q_t preserves context-dependent local changes. A learned router fuses them as $\xi_t = \psi_\Theta(p_t, q_t, r_t)$, combining Persistent and transient evidence before state evolution. Thus, a_t describes what happened, r_t specifies its runtime context, and ξ_t summarizes both temporal evidence streams, allowing the same event to induce different updates under different contexts.

The fused evidence is propagated through L stacked dynamics layers. Set $h_t^{(0)} = W_\xi \xi_t + b_\xi \in \mathbb{R}^d$. At layer ℓ, the controller gives $(\delta_t^{(\ell)}, B_t^{(\ell)}, \gamma_t^{(\ell)}) = \Omega_\Theta^{(\ell)}(h_t^{(\ell-1)}, r_t)$, where $h_t^{(\ell)}$, $\delta_t^{(\ell)}$, and $\gamma_t^{(\ell)}$ are in $\mathbb{R}^d$ and $B_t^{(\ell)} \in \mathbb{R}^{d\times d}$. Define $\Delta_t^{(\ell)} = \mathrm{diag}(\mathrm{softplus}(\delta_t^{(\ell)}))$, $A^{(\ell)} = -\mathrm{diag}(\exp(a^{(\ell)}))$, and $g_t^{(\ell)} = \sigma(\gamma_t^{(\ell)})$. The update is

$$\begin{aligned} h_t^{(\ell)} = \underbrace{\exp(\Delta_t^{(\ell)} A^{(\ell)})}_{\bar{A}_t^{(\ell)}} h_{t-1}^{(\ell)} + \underbrace{\left[\int_0^1 \exp((1-s)\Delta_t^{(\ell)} A^{(\ell)})\, ds\right] \Delta_t^{(\ell)} B_t^{(\ell)} h_t^{(\ell-1)}}_{\bar{B}_t^{(\ell)} h_t^{(\ell-1)}} \\ + \underbrace{g_t^{(\ell)} \odot \mathcal{R}_\Theta^{(\ell)}(h_t^{(\ell-1)})}_{\text{context-gated residual}}, \qquad \ell = 1, \ldots, L. \end{aligned} \tag{4}$$

Here $\bar{A}_t^{(\ell)}, \bar{B}_t^{(\ell)} \in \mathbb{R}^{d\times d}$ and each residual/state is in $\mathbb{R}^d$; the final trajectory is $h_t = h_t^{(L)}$. The three operations respectively control memory retention, evidence injection, and the context-gated

residual at each layer. The recurrence is evaluated sequentially in t and then composed across layers. Consequently, $\bar{A}_t^{(\ell)}$ carries forward the previous state, $\bar{B}_t^{(\ell)} h_t^{(\ell-1)}$ injects the current-layer evidence, and the gated residual supplies a local correction whose strength is determined by the calibrated context. Because the transition matrices are diagonalized through $A^{(\ell)}$ and the step sizes are positive, each layer has an explicit retention scale while remaining compatible with the selective, input-dependent update.

3.4 Behavioral Transition Deviation

After the latent state transitions, let $H = [h_1; \ldots; h_T] \in \mathbb{R}^{T \times d}$. A learned Deviation Probe maps the complete latent state trajectory of one log window to a single logit, $z_{\text{win}} = \text{DP}(H_{1:T})$, and hence to the window-level probability $\widehat{y}_{\text{win}} = \sigma(z_{\text{win}})$. Since $H_{1:T}$ is generated by the latent state transitions conditioned on runtime context in Eq. (3), the readout identifies the behavioral transition deviation from the latent state trajectory. Normal and anomalous path patterns are learned directly from labeled windows.

For a dataset of N labeled windows, $\mathcal{D}_{\text{win}} = \{(\widetilde{\mathcal{O}}_{1:T}^{(i)}, y_{\text{win}}^{(i)})\}_{i=1}^{N}$, all parameters are optimized end to end with one loss term per complete window,

$$\mathcal{L}_{\text{win}}(\Theta) = -\frac{1}{N} \sum_{i=1}^{N} \left[y_{\text{win}}^{(i)} \log \widehat{y}_{\text{win}}^{(i)} + \left(1 - y_{\text{win}}^{(i)}\right) \log\left(1 - \widehat{y}_{\text{win}}^{(i)}\right) \right]. \tag{5}$$

Each term therefore compares one window label with the prediction obtained from all T events in that window; no event-level labels or annotated latent state transition targets are required. The resulting model learns discriminative behavioral trajectories from noisy observations and imperfect parsing.

4 Experiments

4.1 Datasets

We evaluate PulseLog on five labeled datasets from LogHub (Jiang et al., 2024). HDFS_v1 contains 11M log messages collected from benchmark workloads in a private-cloud environment based on the original Amazon EC2 traces, with 53 error types (Xu et al., 2009). HDFS_v3 (TraceBench) contains 15M HDFS log messages collected from a real IaaS deployment using MTracer, covering 17 injected functional and performance faults (Zhou et al., 2014). Hadoop contains 394K MapReduce log messages collected from a five-machine, 46-core cluster running WordCount and PageRank under machine, network, and disk failures (Lin et al., 2016). BGL contains 5M log messages collected from the Blue Gene/L supercomputer at Lawrence Livermore National Laboratory, with 143 error types(Oliner & Stearley, 2007). Thunderbird contains 10M log messages collected from the Thunderbird supercomputer at Sandia National Laboratories, with 95 error types (Oliner & Stearley, 2007).

4.2 Implementation Details

Experimental Settings: PulseLog is implemented in PyTorch 2.9.1 and evaluated on a server with an Intel Xeon Platinum 8558 CPU and eight NVIDIA RTX 4090 GPUs running Ubuntu 22.04.5. All methods use the same preprocessing, data splits, and window construction. Each window contains 20 log events represented by 768-dimensional embeddings, and the batch size is 64. Unless otherwise stated, PulseLog uses a hidden dimension of 256, two stacked selective dynamics layers, and a persistent-to-transient temporal-scale ratio of 2. It is trained with Adam (initial learning rate 10^{-2}, $(\beta_1, \beta_2) = (0.9, 0.999)$, and $\epsilon = 10^{-8}$) and OneCycleLR for 50 epochs. We run PulseLog with random seeds 42, 43, and 44 and report the mean results across the three runs. Deterministic cuDNN is enabled.

Baseline Methods: We compare PulseLog with fifteen baselines, including thirteen public log anomaly detection methods and two LLM-based baselines, GPT-4o (Hurst et al., 2024) and DeepSeek-Flash (Xu et al., 2026). The thirteen public methods are PCA (He et al., 2017a), KNN

(Yu et al., 2024), GAT (Veličković et al., 2017), DeepLog (Du et al., 2017), LogAnomaly (Meng et al., 2019), LogRobust (Zhang et al., 2019), PLELog (Yang et al., 2021), NeuralLog (Le & Zhang, 2021), LogFormer (Guo et al., 2024), LogSD (Xie et al., 2024), LogDLR (Zhou et al., 2025), Lograph (Chu et al., 2025), and MIDLog (He et al., 2025). The baselines cover statistical, graph-based, sequential, semantic, self-supervised, and LLM-based approaches. All methods are evaluated under the same data splits, preprocessing, labels, and evaluation protocols.

Evaluation Metrics: For window-level anomaly detection, anomalous windows are treated as the positive class. We report Precision, Recall, and F1, along with trainable parameters (Params), FLOPs and MACs per forward pass, and serialized model size to assess computational efficiency under identical input dimensions.

4.3 Comparison with SOTA Methods

Table 1: Performance comparison on five log anomaly detection datasets. **Blue** and Orange backgrounds denote the best and second-best results, respectively.

Method	HDFS_v1			HDFS_v3			BGL			Hadoop			Thunderbird		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Deeplog	0.9840	0.7865	0.8743	0.9985	0.8820	0.9366	0.9100	0.9764	0.9420	0.8527	0.9935	0.9177	0.9999	**1.0000**	**1.0000**
GAT	0.9836	0.7779	0.8687	0.9890	0.8746	0.9283	0.9252	0.9720	0.9481	0.8579	0.9968	0.9221	0.9999	**1.0000**	**1.0000**
PCA	0.3080	0.6865	0.4253	0.6768	0.8637	0.7589	0.1238	**0.9864**	0.2200	0.8520	0.7306	0.7866	0.4664	0.7173	0.5652
Loganomaly	0.9831	0.7781	0.8687	0.9969	0.8766	0.9329	0.9819	0.9744	0.9781	0.8560	0.9935	0.9197	0.9999	**1.0000**	**1.0000**
Robustlog	0.9218	0.7592	0.8326	0.9962	0.8761	0.9323	0.9882	0.9740	0.9811	0.8570	0.9935	0.9202	0.9998	0.9999	0.9999
PLELog	0.9797	0.7822	0.8699	0.9979	0.8778	0.9340	0.9760	0.9740	0.9750	0.8571	0.9981	0.9222	0.9999	**1.0000**	**1.0000**
Neurallog	0.8679	0.7756	0.8192	0.9960	0.7871	0.8793	0.9962	0.9680	0.9819	0.8270	**1.0000**	0.9053	0.9998	0.9998	0.9998
KNN	0.8416	0.7432	0.7892	0.9138	0.8496	0.8805	0.9516	0.9098	0.9302	0.9427	0.8618	0.9004	0.9942	0.9785	0.9863
Logformer	0.9699	0.6555	0.7823	0.7612	0.6724	0.7140	0.9968	0.7267	0.8406	0.8270	**1.0000**	0.9053	0.9986	0.9991	0.9988
LogSD	**0.9852**	0.7828	0.8724	0.9975	0.8816	0.9360	0.9932	0.9787	0.9859	0.8523	0.9942	0.9178	0.9999	0.9999	0.9999
Deepseek	0.6818	**1.0000**	0.8108	0.7136	0.7948	0.7510	0.6673	0.5958	0.6297	0.7483	0.6759	0.7099	0.9068	0.8607	0.8821
GPT	0.7926	0.9235	0.8532	0.7618	0.8427	0.7992	0.7786	0.7217	0.7481	0.7924	0.7486	0.7699	0.8963	0.8457	0.8703
LogDLR	0.8913	0.8011	0.8438	0.9983	0.8818	0.9364	0.9953	0.9770	0.9861	0.8634	0.9864	0.9208	0.9999	0.9999	0.9999
Lograph	0.9830	0.7865	0.8739	0.9975	0.8806	0.9354	0.9460	0.9740	0.9598	0.8547	0.9909	0.9178	0.9999	0.9999	0.9999
MIDLog	0.7785	0.9841	0.8693	0.8815	**0.9977**	0.9360	0.9747	0.9225	0.9479	0.8535	0.9922	0.9176	0.9997	0.9995	0.9996
PulseLog	0.8457	0.9637	**0.9009**	**0.9991**	0.8818	**0.9368**	**0.9997**	0.9744	**0.9869**	**0.8681**	0.9935	**0.9266**	**1.0000**	**1.0000**	**1.0000**

Standard Baseline Comparison. Table 1 shows that PulseLog achieves the best F1 on all five datasets, while the margin varies with the operating characteristics of each system. HDFS_v1 contains highly imbalanced block traces from benchmark workloads. PulseLog reaches 90.09% F1 and 96.37% recall, improving over the strongest competing F1 by 2.66 percentage points and maintaining broad anomaly coverage. This result is consistent with retaining transient failure evidence alongside the persistent execution pattern when local cues are noisy or sparse.

Table 2: Complexity comparison on BGL dataset. "-" denotes that the corresponding quantity is not applicable.

Method	Year	Params	FLOPs	MACs	Size
DeepLog	CCS 2017	0.25M	9.91M	4.96M	2.83MB
GAT	ICLR 2018	0.26M	10.49M	5.24M	3.03MB
PCA	TDSC 2018	-	-	-	13.4MB
LogAnomaly	IJCAI 2019	0.82M	31.87M	15.94M	9.45MB
LogRobust	FSE 2019	0.35M	13.83M	6.91M	3.96MB
PLELog	ICSE 2021	0.72M	28.29M	14.14M	8.30MB
NeuralLog	ASE 2021	9.85M	15.73M	7.87M	88.7MB
KNN	ICSE 2024	-	-	-	172MB
LogFormer	AAAI 2024	28.87M	1.45G	0.57G	165MB
LogSD	FSE 2024	0.98M	35.59M	17.79M	11.2MB
LogDLR	TDSC 2025	59.99M	31.72M	15.86M	458MB
Lograph	TSE 2025	1.34M	52.76M	26.38M	15.3MB
MIDLog	ICSE 2025	1.38M	54.14M	27.07M	15.8MB
PulseLog	ICLR 2027	1.20M	41.43M	20.72M	13.8MB

HDFS_v3 (TraceBench) combines different cluster scales, workloads, and 17 functional or performance faults. PulseLog obtains 93.68% F1 and the highest precision (99.91%), consistent with stable discrimination when similar event semantics occur under different runtime conditions. Hadoop provides a complementary case: its WordCount and PageRank traces include machine, network, and disk failures. PulseLog obtains 92.66% F1 with 86.81% precision and 99.35% recall. This precision–recall profile is consistent with using runtime context to distinguish failure-related transitions from benign job variation while preserving anomaly coverage.

BGL and Thunderbird are long-running supercomputer streams with strong regularities and localized alert deviations. PulseLog obtains 98.69% F1 on BGL, 0.08 percentage points above LogDLR, and reaches 100% F1 on Thunderbird after rounding. Because Thunderbird is near saturation for the strongest sequence methods, this result mainly indicates that the proposed representation preserves high accuracy on a large, regular log stream. Across the five datasets, the comparisons sup-

port the motivation for combining Persistent evidence and transient evidence with runtime-context-conditioned latent state transitions; the size of the improvement depends on the operating characteristics of each system. PulseLog also outperforms the two LLM-based baselines on every dataset.

Table 2 shows that PulseLog remains lightweight, with 1.20M parameters, 41.43M FLOPs, 20.72M MACs, and a 13.8MB model size. Although larger than the simplest baselines, it is substantially smaller than high-capacity models such as LogFormer and LogDLR, while achieving competitive accuracy with moderate computational and storage costs.

Table 3: Cross-dataset anomaly detection results.

Method	HDFS_v1 → HDFS_v3			HDFS_v1 → BGL			Hadoop → BGL		
	P	R	F1	P	R	F1	P	R	F1
DeepLog	0.9926±0.0010	0.8796±0.0042	0.9327±0.0028	0.9171±0.0035	0.9690±0.0024	0.9424±0.0021	0.9258±0.0032	0.9587±0.0028	0.9419±0.0024
GAT	0.9654±0.0032	0.8503±0.0061	0.9042±0.0048	0.9871±0.0019	0.9674±0.0025	0.9771±0.0017	0.9814±0.0022	0.9628±0.0029	0.9719±0.0023
PCA	0.6942±0.0084	0.5837±0.0102	0.6342±0.0091	0.6185±0.0096	0.7316±0.0083	0.6703±0.0088	0.6478±0.0089	0.7684±0.0076	0.7029±0.0081
LogAnomaly	0.9913±0.0008	0.5590±0.0126	0.7149±0.0104	0.9296±0.0038	0.9760±0.0020	0.9523±0.0026	0.9362±0.0035	0.9671±0.0026	0.9514±0.0028
RobustLog	0.9828±0.0012	0.8871±0.0041	0.9325±0.0029	0.9012±0.0053	0.9654±0.0028	0.9322±0.0037	0.8947±0.0058	0.9583±0.0032	0.9254±0.0042
PLELog	0.9891±0.0009	0.8856±0.0039	0.9345±0.0027	0.9786±0.0024	0.9734±0.0022	0.9760±0.0019	0.9728±0.0026	0.9657±0.0025	0.9692±0.0022
NeuralLog	0.9598±0.0047	0.6050±0.0108	0.7422±0.0083	0.9945±0.0010	0.9544±0.0030	0.9740±0.0021	**0.9892±0.0014**	0.9618±0.0027	0.9753±0.0019
KNN	0.8214±0.0072	0.6843±0.0094	0.7466±0.0085	0.7628±0.0081	0.8215±0.0073	0.7901±0.0077	0.7816±0.0078	0.8423±0.0069	0.8108±0.0073
LogFormer	0.9874±0.0015	0.8780±0.0043	0.9295±0.0031	0.9847±0.0020	0.9542±0.0031	0.9692±0.0024	0.9798±0.0024	0.9586±0.0029	0.9691±0.0025
LogSD	0.9911±0.0008	0.8851±0.0038	0.9351±0.0026	0.9831±0.0022	0.9694±0.0025	0.9762±0.0019	0.9785±0.0025	0.9637±0.0028	0.9710±0.0024
LogDLR	0.9932±0.0008	0.8841±0.0037	0.9355±0.0025	**0.9969±0.0006**	0.9634±0.0027	0.9799±0.0016	0.9885±0.0017	0.9724±0.0024	0.9804±0.0018
Lograph	0.9817±0.0012	**0.8875±0.0041**	0.9322±0.0029	0.8415±0.0064	0.9581±0.0030	0.8960±0.0052	0.8627±0.0059	0.9446±0.0035	0.9018±0.0048
MIDLog	0.9884±0.0011	0.8858±0.0039	0.9343±0.0028	0.9684±0.0028	0.9168±0.0038	0.9419±0.0032	0.9734±0.0026	0.9239±0.0036	0.9480±0.0030
PulseLog	**0.9958±0.0005**	0.8824±0.0038	**0.9357±0.0025**	0.9841±0.0020	**0.9792±0.0018**	**0.9816±0.0014**	0.9876±0.0017	**0.9850±0.0016**	**0.9863±0.0012**

Few-Shot Cross-Dataset Adaptation. Table 3 evaluates cross-dataset adaptation with fine-tuning on 20K training samples from the target system. HDFS_v1 → HDFS_v3 uses HDFS_v1 as the source and HDFS_v3 as the target; the other settings follow the same protocol.

The results show that PulseLog maintains a balanced precision–recall profile with limited target-domain adaptation. It achieves 93.57% F1 and 99.58% precision on HDFS_v1 → HDFS_v3, 98.16% F1 and 97.92% recall on HDFS_v1 → BGL, and 98.63% F1 and 98.50% recall on Hadoop → BGL. Despite differences in event semantics, workloads, and fault manifestations across HDFS, Hadoop, and Blue Gene/L, PulseLog maintains strong anomaly detection performance across transfer settings, supporting the effectiveness of context-conditioned latent state transitions and Multi-Rate Behavioral Signatures with limited target-domain data.

4.4 Robustness and Data Efficiency Evaluation

We construct 100K-instance stress-test subsets from BGL, HDFS_v1, Hadoop, and Thunderbird, preserving their original log templates, error types, and an anomaly ratio of approximately 2.7%. This controls data scale and class imbalance while retaining system-specific diversity, enabling evaluation under limited training data and perturbed local log cues.

Limited Training Data. We evaluate data efficiency on the 100K-instance subsets described above. For each dataset, 80K instances are fixed as the test set, while the training set is varied from 5K to 20K instances. This protocol keeps the test distribution and class imbalance fixed, allowing us to isolate the effect of training-set size.

Table 4 shows that PulseLog achieves the highest F1 across all datasets and training budgets. Its F1 remains above 96% on BGL, while HDFS and Hadoop improve from 98.68% to 99.51% and from 97.74% to 99.77%, respectively. Thunderbird reaches 100% F1 with only 5K samples, leaving little room for further gains.

These scaling patterns reflect differences in execution-pattern diversity. Stable operating regularities in BGL and Thunderbird enable effective learning with limited supervision, whereas the more varied workload- and context-dependent patterns in HDFS and Hadoop benefit from additional data. PulseLog's Persistent and transient evidence, together with runtime-context-conditioned state transitions, allows it to capture stable behavioral structure while incorporating additional variation as training data grows.

Robustness to Noise. We evaluate noise robustness on BGL by using an LLM to inject context-aware perturbations into original logs before training and testing. The LLM infers template structure, parameter types, and contextual constraints, then applies targeted perturbations while preserving the original format. As shown in Table 5, the Controlled Noise Injection process introduces template,

Table 4: F1 scores under different training-set sizes. The best and second-best results in each row are highlighted in blue and orange, respectively.

		DeepLog	GAT	KNN	LogAnomaly	LogDLR	LogFormer	Lograph	NeuralLog	PCA	PLELog	LogRobust	LogSD	MIDLog	PulseLog
BGL	5K	0.5983	0.8166	0.5166	0.7107	0.9604	0.7562	0.6513	0.7121	0.7184	0.9206	0.7784	0.9002	0.7430	**0.9630**
	10K	0.8122	0.9606	0.5160	0.7497	0.9376	0.7931	0.9397	0.7526	0.7754	0.8821	0.7877	0.9009	0.8689	**0.9644**
	15K	0.8161	0.9720	0.3502	0.8320	0.9164	0.8259	0.9129	0.7795	0.7778	0.9381	0.7778	0.9042	0.8618	**0.9738**
	20K	0.8166	0.9562	0.5787	0.8741	0.9298	0.8447	0.8951	0.8036	0.7778	0.9502	0.8225	0.9030	0.8715	**0.9615**
HDFS	5K	0.5993	0.9248	0.0552	0.8132	0.9846	0.9027	0.9557	0.8638	0.5850	0.9854	0.9556	0.8769	0.9654	**0.9868**
	10K	0.9209	0.9671	0.1302	0.9101	0.9887	0.9316	0.9850	0.8909	0.5859	0.9898	0.9671	0.9445	0.9903	**0.9912**
	15K	0.9431	0.9831	0.1457	0.9770	0.9915	0.9578	0.9912	0.9165	0.6033	0.9928	0.9788	0.9744	0.9930	**0.9934**
	20K	0.9605	0.9893	0.1676	0.9868	0.9934	0.9824	0.9937	0.9412	0.6641	0.9944	0.9825	0.9898	0.9946	**0.9951**
Hadoop	5K	0.6889	0.9591	0.5762	0.9638	0.9698	0.8743	0.9569	0.7348	0.5982	0.9718	0.8844	0.9700	0.7787	**0.9774**
	10K	0.9624	0.9875	0.5959	0.8047	0.9916	0.9068	0.9586	0.7816	0.5847	0.9923	0.9422	0.9915	0.9440	**0.9935**
	15K	0.9755	0.9883	0.5501	0.9916	0.9925	0.9345	0.9903	0.8463	0.6125	0.9930	0.9535	0.9926	0.9858	**0.9949**
	20K	0.9873	0.9926	0.5182	0.9932	0.9935	0.9617	0.9915	0.9808	0.6018	0.9940	0.9558	0.9929	0.9897	**0.9977**
Thunderbird	5K	0.8403	**1.0000**	0.5931	**1.0000**	**1.0000**	0.8631	0.9088	0.8383	0.5959	**1.0000**	0.8665	0.9914	0.8271	**1.0000**
	10K	0.9864	**1.0000**	0.5708	**1.0000**	**1.0000**	0.9198	0.9965	0.8991	0.6037	**1.0000**	0.9398	**1.0000**	0.9979	**1.0000**
	15K	0.9926	**1.0000**	0.5623	**1.0000**	**1.0000**	0.9479	**1.0000**	0.9297	0.5874	**1.0000**	0.9460	**1.0000**	**1.0000**	**1.0000**
	20K	0.9985	**1.0000**	0.5425	**1.0000**	**1.0000**	0.9676	**1.0000**	0.9545	0.6106	0.9994	0.9485	0.9997	**1.0000**	**1.0000**

parameter, token, and value-level corruption while retaining the underlying log structure. This setup tests whether PulseLog can remain robust when local semantic and contextual cues are perturbed.

Table 6 shows that PulseLog achieves the highest F1 at all three noise levels, reaching 95.08%, 87.23%, and 82.51% under 1%, 2%, and 5% noise, respectively. At 2% noise, it maintains 89.52% precision and 85.06% recall, compared with 82.34% F1 for the strongest baseline. Under 5% noise, PulseLog retains 100.00% precision with 70.22% recall, indicating increasingly conservative anomaly decisions as local observations become less reliable.

Table 5: Illustration of controlled noise injection into raw logs.

Controlled Noise Injection	
Input	`INFO: Received block blk_1073741825 from 10.0.0.12`
Inference	Infer the template, parameter types, and contextual constraints.
Perturbation	`INFO` → `WARNING` `10.0.0.12` → `10.0.0.999`
Validation	Preserve the log format and verify contextual inconsistency.
Output	`WARNING: Received block blk_1073741825 from 10.0.0.999`

From 1% to 5% noise, PulseLog's F1 drops by 12.57 percentage points, compared with 15.06 for LogSD, 17.26 for Lograph, and 39.84 for DeepLog. This stability reflects its reliance on behavioral state transitions rather than isolated template semantics: Persistent evidence anchors the execution regime, while transient evidence captures local deviations, with runtime context conditioning their integration. Since the perturbations are synthetic, the results demonstrate robustness under the tested noise conditions rather than all forms of real-world log corruption.

4.5 Ablation and Representation Analysis

We conduct our analysis on the 100K-instance stress-test subsets introduced in the previous section.

Figure 2 illustrates why PulseLog models anomalies as context-conditioned behavioral transition deviations rather than relying on isolated log observations. Raw features show substantial overlap between normal and anomalous windows, particularly in PCA, while t-SNE reveals mixed local clusters, indicating that noisy observations alone provide limited separation. After PulseLog constructs behavioral features, the two classes become more distinguishable, with normal windows forming more compact regions and anomalous windows occupying more localized clusters. The

Table 6: Performance under different noise levels on BGL.

Method	1%			2%			5%		
	P	R	F1	P	R	F1	P	R	F1
DeepLog	0.9939±0.0021	0.8814±0.0052	0.9343±0.0038	0.6122±0.0142	0.8489±0.0108	0.7113±0.0121	0.6608±0.0215	0.4507±0.0182	0.5359±0.0194
GAT	0.9948±0.0018	0.8941±0.0048	0.9417±0.0035	0.6397±0.0128	0.9785±0.0065	0.7736±0.0103	0.9971±0.0010	0.5885±0.0201	0.7399±0.0168
PCA	0.2156±0.0102	**0.9934±0.0010**	0.3543±0.0128	0.5378±0.0195	**1.0000±0.0000**	0.6994±0.0172	0.2487±0.0148	**1.0000±0.0000**	0.3984±0.0161
LogAnomaly	0.9456±0.0032	0.8623±0.0058	0.9010±0.0047	0.7972±0.0105	0.5940±0.0140	0.6808±0.0122	0.9064±0.0068	0.5453±0.0176	0.6809±0.0148
RobustLog	0.9906±0.0024	0.8216±0.0068	0.8982±0.0051	0.7566±0.0136	0.6503±0.0157	0.6994±0.0141	0.9807±0.0032	0.6137±0.0185	0.7550±0.0147
PLELog	0.9925±0.0020	0.8920±0.0049	0.9396±0.0037	0.8228±0.0094	0.7346±0.0117	0.7762±0.0102	0.9855±0.0028	0.6841±0.0159	0.8076±0.0125
NeuralLog	0.9475±0.0038	0.9158±0.0047	0.9312±0.0041	0.8756±0.0073	0.7767±0.0101	0.8234±0.0085	0.8427±0.0092	0.7634±0.0120	0.8015±0.0101
KNN	0.5012±0.0086	0.6419±0.0092	0.5629±0.0088	0.4862±0.0172	0.8032±0.0125	0.6057±0.0151	0.3540±0.0224	0.5755±0.0196	0.4383±0.0211
LogFormer	0.8426±0.0051	0.7315±0.0076	0.7831±0.0062	0.8018±0.0096	0.6824±0.0124	0.7374±0.0108	0.7645±0.0138	0.6358±0.0162	0.6942±0.0149
LogSD	0.9980±0.0009	0.8952±0.0045	0.9438±0.0032	0.7950±0.0101	0.6678±0.0138	0.7259±0.0118	0.9939±0.0022	0.6600±0.0169	0.7932±0.0130
LogDLR	0.9415±0.0035	0.8126±0.0069	0.8724±0.0054	**0.9631±0.0038**	0.5958±0.0135	0.7362±0.0102	0.9970±0.0011	0.6761±0.0155	0.8058±0.0121
Lograph	**1.0000±0.0000**	0.8982±0.0046	0.9464±0.0031	0.8300±0.0098	0.6520±0.0131	0.7303±0.0112	0.9785±0.0038	0.6398±0.0171	0.7737±0.0139
MIDLog	0.8964±0.0041	0.9836±0.0022	0.9380±0.0034	0.7610±0.0112	0.6661±0.0139	0.7104±0.0120	0.8525±0.0088	0.3139±0.0162	0.4588±0.0151
PulseLog	0.9921±0.0007	0.9128±0.0032	**0.9508±0.0018**	0.8952±0.0032	0.8506±0.0040	**0.8723±0.0035**	**1.0000±0.0000**	0.7022±0.0068	**0.8251±0.0054**

ablations further reveal the complementary roles of the proposed components. Removing transient evidence produces the largest F1 degradation of 2.82, 3.09, and 3.05 percentage points on HDFS, BGL, and Hadoop, respectively, highlighting the importance of short-lived behavioral deviations. Removing Multi-Rate Fusion decreases F1 by 1.85–2.02 percentage points, supporting the benefit of combining Persistent and transient evidence. Removing the residual path and Deviation Probe reduces F1 by 1.22–1.47 and 1.96–2.22 percentage points, respectively. Together, these results support the effectiveness of behavioral representations and the complementary contributions of the proposed components to context-conditioned state modeling and behavioral transition deviation detection.

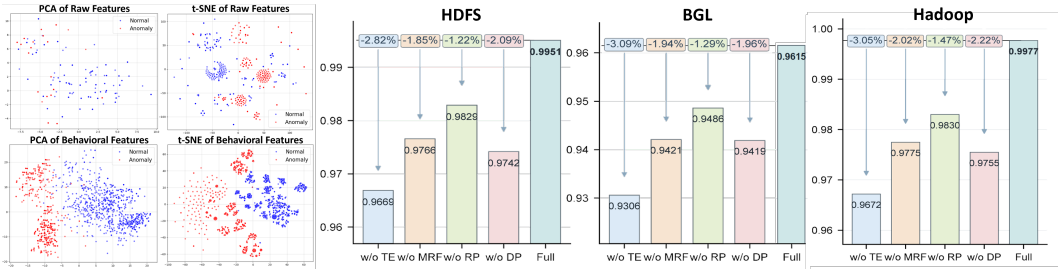

Figure 2: Representation visualization and ablation study of PulseLog. The left panel visualizes the representations of normal and anomalous samples, obtained by normalized weighted aggregation across three datasets, for both normal and anomaly features. The right panel reports F1 scores for different PulseLog variants. TE denotes transient evidence, MRF denotes Multi-Rate Fusion, RP denotes the residual path, and DP denotes the Deviation Probe.

5 Conclusion

Log anomaly detection must infer system behavior from observations affected by template and parameter noise. It must also distinguish Persistent from transient evidence and account for runtime context. We address this challenge by formulating anomaly detection as context-conditioned behavioral state dynamics. We propose PulseLog, which constructs Multi-Rate Behavioral Signatures, evolves latent state transitions under runtime context, and detects window-level anomalies through behavioral transition deviations. Across five public log systems, PulseLog achieves strong performance under the standard protocol. It remains effective under observation noise, reduced training data, and cross-system adaptation. It also maintains a compact model footprint. These results support the central premise of PulseLog: robust detection benefits from modeling how system behavior evolves under runtime context. It is preferable to treating isolated log events as equally reliable or decisive evidence. Future work will further evaluate PulseLog on real-world production systems. This will assess its effectiveness under naturally occurring noise and operational variability.

AI Assistance Disclosure

We used generative AI tools for limited assistance with manuscript editing, perturbed log data generation, and baseline implementation and evaluation. The authors reviewed and verified all data, code, experiments, analyses, and results, and take full responsibility for the final content and claims.

REFERENCES

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Çağlar Gulçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1724–1734, 2014.

Guojun Chu, Jingyu Wang, Qi Qi, Haifeng Sun, Zirui Zhuang, Bo He, Yuhan Jing, Lei Zhang, and Jianxin Liao. Anomaly detection on interleaved log data with semantic association mining on log-entity graph. *IEEE Transactions on Software Engineering*, 51(2):581–594, 2025.

Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan Mathur, Rajat Sen, and Rose Yu. Long-term forecasting with tide: Time-series dense encoder. *arXiv preprint arXiv:2304.08424*, 2023.

Min Du, Feifei Li, Guineng Zheng, and Vivek 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*, pp. 1285–1298, 2017.

Mononito Goswami, Konrad Szafer, Arjun Choudhry, Yifu Cai, Shuo Li, and Artur Dubrawski. Moment: A family of open time-series foundation models. *arXiv preprint arXiv:2402.03885*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Haixuan Guo, Shuhan Yuan, and Xintao Wu. Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2021.

Hongcheng Guo, Jian Yang, Jiaheng Liu, Jiaqi Bai, Boyang Wang, Zhoujun Li, Tieqiao Zheng, Bo Zhang, Junran Peng, and Qi Tian. Logformer: A pre-train and tuning pipeline for log anomaly detection. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 135–143, 2024.

Minghua He, Tong Jia, Chiming Duan, Huaqian Cai, Ying Li, and Gang Huang. Weakly-supervised log-based anomaly detection with inexact labels via multi-instance learning. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 2918–2930. IEEE, 2025.

Pinjia He, Jieming Zhu, Shilin He, Jian Li, and Michael R Lyu. Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing*, 15(6): 931–944, 2017a.

Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, pp. 33–40. IEEE, 2017b.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R Lyu. A large-scale evaluation for log parsing techniques: How far are we? In *Proceedings of the 33rd ACM SIGSOFT international symposium on software testing and analysis*, pp. 223–234, 2024.

Ming Jin, Shiyu Wang, Lintao Ma, Zhixuan Chu, James Zhang, Xiaoming Shi, Pin-Yu Chen, Yuxuan Liang, Yuan-Fang Li, Shirui Pan, et al. Time-llm: Time series forecasting by reprogramming large language models. In *International conference on learning representations*, volume 2024, pp. 23857–23880, 2024.

Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel C Briand. Impact of log parsing on deep learning-based anomaly detection. *Empirical Software Engineering*, 29(6):139, 2024.

Max Landauer, Sebastian Onder, Florian Skopik, and Markus Wurzenberger. Deep learning for anomaly detection in log data: A survey. *Machine Learning with Applications*, 12:100470, 2023.

Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 492–504. IEEE, 2021.

Van-Hoang Le and Hongyu Zhang. Prelog: A pre-trained model for log analytics. *Proceedings of the ACM on Management of Data*, 2(3):1–28, 2024.

Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th international conference on software engineering companion*, pp. 102–111, 2016.

Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting. In *International conference on learning representations*, volume 2024, pp. 11116–11140, 2024a.

Yong Liu, Haoran Zhang, Chenyu Li, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer: Generative pre-trained transformers are large time series models. *arXiv preprint arXiv:2402.02368*, 2024b.

Donghao Luo and Xue Wang. Moderntcn: A modern pure convolution structure for general time series analysis. In *International conference on learning representations*, volume 2024, pp. 31728–31770, 2024.

Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Ijcai*, volume 19, pp. 4739–4745, 2019.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

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

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 5100–5111, 2019.

Yongliang Tao, Hongyu Zhang, Van-Hoang Le, and Yi Xiao. Knowledge-augmented log anomaly detection with large language models. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, pp. 131–143, 2026.

Shreshth Tuli, Giuliano Casale, and Nicholas R Jennings. Tranad: Deep transformer networks for anomaly detection in multivariate time series data. *arXiv preprint arXiv:2201.07284*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Shiyu Wang, Haixu Wu, Xiaoming Shi, Tengge Hu, Huakun Luo, Lintao Ma, James Zhang, and Jun Zhou. Timemixer: Decomposable multiscale mixing for time series forecasting. In *International conference on learning representations*, volume 2024, pp. 38626–38652, 2024.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. *arXiv preprint arXiv:2210.02186*, 2022.

Yongzheng Xie, Hongyu Zhang, and Muhammad Ali Babar. Loggd: Detecting anomalies from system logs with graph neural networks. In *2022 IEEE 22nd International conference on software quality, reliability and security (QRS)*, pp. 299–310. IEEE, 2022.

Yongzheng Xie, Hongyu Zhang, and Muhammad Ali Babar. Logsd: Detecting anomalies from system logs through self-supervised learning and frequency-based masking. *Proceedings of the ACM on Software Engineering*, 1(FSE):2098–2120, 2024.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Jiehui Xu, Haixu Wu, Jianmin Wang, and Mingsheng Long. Anomaly transformer: Time series anomaly detection with association discrepancy. *arXiv preprint arXiv:2110.02642*, 2021.

Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 117–132, 2009.

Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 1448–1460. IEEE, 2021.

Yiyuan Yang, Chaoli Zhang, Tian Zhou, Qingsong Wen, and Liang Sun. Dcdetector: Dual attention contrastive representation learning for time series anomaly detection. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 3033–3045, 2023.

Boxi Yu, Jiayi Yao, Qiuai Fu, Zhiqing Zhong, Haotian Xie, Yaoliang Wu, Yuchi Ma, and Pinjia He. Deep learning or classical machine learning? an empirical study on log-based anomaly detection. In *Proceedings of the 46th IEEE/ACM international conference on software engineering*, pp. 1–13, 2024.

Zhihan Yue, Yujing Wang, Juanyong Duan, Tianmeng Yang, Congrui Huang, Yunhai Tong, and Bixiong Xu. Ts2vec: Towards universal representation of time series. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 8980–8987, 2022.

Chenyangguang Zhang, Tong Jia, Guopeng Shen, Pinyan Zhu, and Ying Li. Metalog: Generalizable cross-system anomaly detection from logs with meta-learning. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–12, 2024.

Ting Zhang, Xin Huang, Wen Zhao, Shaohuang Bian, and Peng Du. Logprompt: A log-based anomaly detection framework using prompts. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2023.

Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pp. 807–817, 2019.

Jingwen Zhou, Zhenbang Chen, Ji Wang, Zibin Zheng, and Michael R Lyu. Trace bench: An open data set for trace-oriented monitoring. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pp. 519–526. IEEE, 2014.

Junwei Zhou, Yijia Qian, Qingtian Zou, Peng Liu, and Jianwen Xiang. Deepsyslog: Deep anomaly detection on syslog using sentence embedding and metadata. *IEEE Transactions on Information Forensics and Security*, 17:3051–3061, 2022.

Junwei Zhou, Shaowen Ying, Shulan Wang, Dongdong Zhao, Jianwen Xiang, Kaitai Liang, and Peng Liu. Logdlr: Unsupervised cross-system log anomaly detection through domain-invariant latent representation. *IEEE Transactions on Dependable and Secure Computing*, 22(4):4456–4471, 2025.

Tian Zhou, Peisong Niu, Liang Sun, Rong Jin, et al. One fits all: Power general time series analysis by pretrained lm. *Advances in neural information processing systems*, 36:43322–43355, 2023.

Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 121–130. Ieee, 2019.

TABLE OF CONTENTS

1 Introduction . . . 1
2 Related Work . . . 2
2.1 Log Anomaly Detection . . . 2
2.2 Robustness to Noise and Limited Labels . . . 3
3 Methodology . . . 3
3.1 Problem Formulation . . . 3
3.2 Behavioral Evidence from Log Observations . . . 4
3.3 Multi-Rate Context-Conditioned State Dynamics . . . 4
3.4 Behavioral Transition Deviation . . . 5
4 Experiments . . . 5
4.1 Datasets . . . 5
4.2 Implementation Details . . . 5
4.3 Comparison with SOTA Methods . . . 6
4.4 Robustness and Data Efficiency Evaluation . . . 7
4.5 Ablation and Representation Analysis . . . 8
5 Conclusion . . . 9
AI Assistance Disclosure . . . 9
A Theoretical Analysis . . . 15
A.1 Setup and Definitions . . . 15
A.2 Unified Trajectory Formulation . . . 16
A.3 Stability of Context-Conditioned Dynamics . . . 16
A.4 What Multi-Rate Aggregation Adds . . . 16
A.5 Behavioral Transition Deviation and Identifiability . . . 17
A.6 Robustness and Classification Margin . . . 17
B Experimental Details . . . 17
B.1 Dataset Descriptions and Statistics . . . 17
B.2 Window Construction and Data Splits . . . 18
B.3 Implementation Details of LLM-Based Methods . . . 19
B.4 LLM-Based Noise Generation Procedure . . . 20
C Additional Experiments . . . 20
C.1 Backbone Comparisons . . . 20
C.2 Fusion Mechanism Comparisons . . . 21
C.3 Trajectory Readout Strategy Comparisons . . . 21
D Visualization Analytics . . . 22
D.1 Normal and Anomalous Features . . . 22
D.2 Raw and Behavioral Features . . . 23
E Sensitivity Analysis . . . 24

E.1 Sensitivity to Window Length . . . 24
E.2 Sensitivity to Hidden Dimension and Model Depth . . . 25
E.3 Sensitivity to the Multi-Rate Scale Ratio . . . 26
F Detailed Results for Main-Text Tables . . . 27
F.1 Detailed Results for Table 1 . . . 27
F.2 Detailed Results for Table 4 . . . 29

A Theoretical Analysis

This section characterizes PulseLog as a context-indexed, multi-rate dynamical system. The analysis has four goals: define the statistical object represented by the model, establish stability of its discretized dynamics, quantify what the two temporal rates add, and relate behavioral transition deviation to a classification margin.

A.1 Setup and Definitions

Let $e_t = f_{\mathrm{SB}}(\widetilde{\tau}_t)$ and $r_t = f_{\mathrm{char}}(\widetilde{a}_t)$ be the semantic and runtime-context embeddings of event t. Thus, the r_t used for runtime context in the main method is the same quantity denoted c_t in some equations; throughout this appendix we use r_t and set $c_t := r_t$ when relating to that notation. We write $o_t = (e_t, r_t)$ and assume $\|e_t\| \le B_e$ and $\|r_t\| \le B_r$. For $m \in \{\mathrm{per}, \mathrm{tr}\}$, define the context-indexed aggregation operator

$$(\mathcal{K}^r_m e)(u) = \int_0^1 K_m(u, v; r) e(v)\, \mathrm{d}v, \qquad \int_0^1 K_m(u, v; r)\, \mathrm{d}v = 1.$$

For a finite window of T events, use $u_t = t/T$ and the normalized discrete operator

$$(\mathcal{K}^r_{m,T} e)_t = \sum_{j=1}^{T} w_{m,tj} e_j, \qquad w_{m,tj} = \frac{\exp\!\left(s_m(r_t, r_j) - \frac{(u_t - u_j)^2}{2\lambda_m^2}\right)}{\sum_{k=1}^{T} \exp\!\left(s_m(r_t, r_k) - \frac{(u_t - u_k)^2}{2\lambda_m^2}\right)},$$

where s_m is a learned context-compatibility score (set to zero when no context-dependent reweighting is used). The Gaussian temporal envelope makes λ_m the bandwidth in normalized window-time units; weights are renormalized at each query, including near the window boundaries. The persistent and transient branches use $\lambda_{\mathrm{per}} > \lambda_{\mathrm{tr}} > 0$. This finite sum is the normalized Riemann-sum implementation of the continuous operator above. The two signatures are

$$p_t = \phi_{\mathrm{per}}((\mathcal{K}^r_{\mathrm{per},T} e)_t, r_t), \qquad q_t = \phi_{\mathrm{tr}}((\mathcal{K}^c_{\mathrm{tr}} e)(u_t), c_t), \qquad \xi_t = \psi(p_t, q_t, r_t, c_t). \tag{A.1}$$

The latent transition for one dynamics layer is the affine time-varying system

$$h_t = M_t h_{t-1} + b_t, \quad M_t = \exp(\Delta_t A), \quad b_t = \bar{B}_t v_t + g_t \odot \mathcal{R}_\Theta(v_t), \tag{A.2}$$

where $v_t := h_t^{(\ell-1)}$ is the current-layer input in main-text Eq. (4), $A = -\operatorname{diag}(\exp a)$, $\Delta_t = \operatorname{diag}(\mathrm{softplus}(\delta_t))$, and $(\delta_t, B_t, \gamma_t) = \Omega_\Theta(v_t, r_t)$. Thus, (A.2) is a single-layer shorthand of the main recurrence: the input term and the gated residual both act on the same current-layer representation v_t; the raw-message variable x_t does not appear in this abstraction. The behavioral transition deviation is the path functional

$$D_\Theta(o_{1:T}) = \mathrm{DP}_\Theta(h_1, \Delta h_2, \ldots, \Delta h_T), \qquad \Delta h_t = h_t - h_{t-1}, \tag{A.3}$$

and $\widehat{y} = \sigma(D_\Theta)$. Thus the learned object is a conditional path distribution $o_{1:T} \mapsto \mathcal{H}_{1:T}$ followed by a scalar functional, not a classifier on independent events.

A.2 Unified Trajectory Formulation

The central design principle of PulseLog is to formulate anomaly detection as a unified trajectory inference problem, where different modeling components jointly operate on a shared context-conditioned behavioral trajectory. Each modeling stage contributes to the construction and evolution of this trajectory, establishing a coherent dependency from observations to the final anomaly decision.

Specifically, semantic representations, runtime context, and multi-rate aggregation jointly characterize the behavioral evidence at each event. This evidence then determines the subsequent state evolution together with the corresponding runtime context. Consequently, the latent state at each step reflects both the accumulated behavioral history and the conditions under which the current event occurs. The resulting trajectory provides the common representation through which behavioral evidence is propagated across the entire window.

This unified dependency can be summarized as

$$o_1; \longrightarrow; \xi_1; \longrightarrow; \mathcal{H}_1; \longrightarrow; \mathcal{H}_1^{\Delta}; \longrightarrow; \widehat{y},$$

where each stage operates on the output of the preceding stage and contributes to the same trajectory-level representation. The multi-rate signatures characterize behavioral evidence at different temporal scales, runtime context conditions how this evidence is incorporated into the state dynamics, and the deviation probe evaluates the resulting state evolution. These elements are therefore coupled through the shared latent trajectory and jointly determine how behavioral patterns are represented, propagated, and assessed.

Thus, PulseLog provides a trajectory-centric formulation in which representation, context conditioning, state evolution, and deviation measurement form successive aspects of a single behavioral process. This formulation also establishes a common structural basis for the component-level ablations reported in the main paper and Appendix C.

A.3 Stability of Context-Conditioned Dynamics

We first state the condition under which the selective discretization is stable.

Theorem 1 (uniform exponential stability). Suppose $0 < \underline{\delta} \le [\Delta_t]_{jj} \le \overline{\delta}$ and $a_j \ge a_{\min} > -\infty$ for every state coordinate. Then

$$\|M_t\|_2 = \max_j \exp(-[\Delta_t]_{jj} e^{a_j}) \le \rho := \exp(-\underline{\delta} e^{a_{\min}}) < 1. \tag{A.4}$$

For bounded forcing $\|b_t\|_2 \le B_b$, the solution of (A.2) satisfies

$$\|h_t\|_2 \le \rho^t \|h_0\|_2 + \frac{1-\rho^t}{1-\rho} B_b, \qquad \|h_t - h_t'\|_2 \le \rho^t \|h_0 - h_0'\|_2. \tag{A.5}$$

Proof. Since A and Δ_t are diagonal, M_t is diagonal with entries $\exp(-[\Delta_t]_{jj} e^{a_j})$, which proves (A.4). Unrolling (A.2), $h_t = (\prod_{i=1}^t M_i) h_0 + \sum_{k=1}^t (\prod_{i=k+1}^t M_i) b_k$. Submultiplicativity gives (A.5); subtracting two solutions gives the second inequality.

The result supplies a quantitative memory horizon. The contribution of an event at time k to h_t is bounded by $\rho^{t-k}\|b_k\|$, so events older than $L_\varepsilon = \lceil \log(\varepsilon/B_b)/\log\rho \rceil$ contribute at most ε to the state. Context changes this horizon through Δ_t and therefore controls retention without changing the stability condition.

A.4 What Multi-Rate Aggregation Adds

For a normalized kernel, define its effective support by $S_m(u;c) = \{v : K_m(u,v;c) \ge \eta\}$. The multi-rate assumption is $|S_{\mathrm{tr}}| < |S_{\mathrm{per}}|$. Let $e = \mu + \varepsilon$ where μ is a piecewise-smooth regime signal and ε is zero-mean local noise.

Theorem 2 (bias–variance separation). If K_m is normalized and $\mathrm{Var}(\varepsilon(v)) \le \sigma^2$ with independent noise across events, then

$$\mathbb{E}[(\mathcal{K}_m^c e)(u)] = (\mathcal{K}_m^c \mu)(u), \qquad \mathrm{tr}\,\mathrm{Cov}[(\mathcal{K}_m^c e)(u)] \le \sigma^2 \|K_m(u,\cdot;c)\|_2^2. \tag{A.6}$$

For approximately uniform support of width λ_m, $\|K_m\|_2^2 = \Theta(1/\lambda_m)$; hence the persistent branch has lower variance, while the transient branch has lower smoothing bias at a localized change. Their joint representation retains both terms in the bias–variance decomposition.

Proof. Linearity gives the expectation. Independence gives the covariance $\int K_m(u,v;c)^2\,\mathrm{Cov}(\varepsilon(v))\,\mathrm{d}v$, whose trace is bounded by the right-hand side. Uniform kernels satisfy $\int K_m^2 = \Theta(1/\lambda_m)$.

This theorem isolates the mechanism tested by the single-rate ablation: p_t estimates the low-variance regime component, while q_t preserves a localized change that a broad kernel attenuates. Context gating makes this decomposition conditional on the operating regime rather than fixed globally.

A.5 Behavioral Transition Deviation and Identifiability

Let $\mathbf{H} = (h_1, \Delta h_2, \ldots, \Delta h_T)$. A trajectory readout is transition-sensitive if there exist two paths with the same multiset of states but different ordered increments and different scores. A pooled readout $G(\frac{1}{T}\sum_t h_t)$ is permutation-invariant and therefore cannot be transition-sensitive.

Proposition 3 (order sensitivity). If DP contains a non-constant temporal operator (e.g., a causal convolution, SSM readout, or attention with positional encoding), then there exist paths $\mathbf{H}, \mathbf{H}'$ with $\{h_t\}_{t=1}^T = \{h'_t\}_{t=1}^T$ as multisets but $D_\Theta(\mathbf{H}) \ne D_\Theta(\mathbf{H}')$. Consequently, behavioral transition deviation can distinguish a benign ordering from an anomalous ordering even when their pooled event statistics coincide.

Proof. Choose two distinct increments $u \ne v$ and exchange their positions. A non-constant temporal operator assigns different coefficients to at least two positions, so its outputs differ by a nonzero multiple of $u - v$. A permutation-invariant pooled readout gives the same output for both paths.

A.6 Robustness and Classification Margin

Assume both trajectories satisfy Theorem 1 with the same contraction bound $\|M_t\|_2, \|M'_t\|_2 \le \rho < 1$, have the same initial state, and have bounded states $\|h'_t\|_2 \le H_*$. Suppose the forcing perturbation is bounded by $\|b_t - b'_t\|_2 \le \epsilon_b$ (including changes to B_t, ξ_t, the gate, and the direct residual), and explicitly assume $\|M_t - M'_t\|_2 \le \epsilon_M$. Then

$$\|h_t - h'_t\|_2 \le \frac{\epsilon_b + \epsilon_M H_*}{1-\rho}, \qquad |D_\Theta - D'_\Theta| \le L_D\sqrt{T}\frac{\epsilon_b + \epsilon_M H_*}{1-\rho}, \tag{A.7}$$

Indeed, subtracting the state recurrences gives $\|h_t - h'_t\|_2 \le \rho\|h_{t-1} - h'_{t-1}\|_2 + \epsilon_M\|h'_{t-1}\|_2 + \epsilon_b$, and summing this geometric recursion yields (A.7). One sufficient choice is $H_* = \|h_0\|_2 + B_b/(1-\rho)$ when both trajectories share h_0 and $\|b'_t\|_2 \le B_b$. Lipschitz continuity of Ω_Θ alone does not imply the required ϵ_M bound; it must be established separately, for example by bounding the map from controller outputs to $\exp(\Delta_t A)$ on a bounded controller domain. If a labeled window has logit margin $\gamma = |D_\Theta|$, its prediction is invariant whenever the right-hand side of (A.7) is smaller than γ. Contraction controls temporal amplification, while the margin measures distance from the decision boundary.

B Experimental Details

B.1 Dataset Descriptions and Statistics

HDFS_v1. HDFS_v1 contains console logs generated by benchmark workloads in a private-cloud Hadoop Distributed File System. The original study groups messages by block identifier and assigns each block trace a binary normal/anomalous label using manually designed rules. The commonly

used release contains approximately 11.2 million log messages, about 575 thousand block traces, and 53 reported error categories; the exact number of windows depends on the windowing rule. LogHub provides the raw log, parsed templates, event traces, anomaly labels, and an event-occurrence matrix. We preserve block-level labels and chronological order rather than assigning labels to templates.

HDFS_v3 (TraceBench). HDFS_v3 is the TraceBench collection obtained with MTracer from an HDFS deployment in a real IaaS environment. It varies cluster scale, request type, workload speed, and execution condition, and contains normal traces together with traces collected under injected faults. The TraceBench description reports more than 180 hours of collection, 364 files, more than 370 thousand traces, and 17 injected functional, performance, and real-system faults. The main-paper message count (approximately 15 million) refers to the expanded log records underlying these traces, whereas the LogHub release exposes trace-level CSV files, event identifiers, and normal/failure task identifiers. We retain trace order and use the provided normal/failure status when constructing windows.

BGL. BGL consists of logs from the Blue Gene/L supercomputer at Lawrence Livermore National Laboratory. The system had 131,072 processors and 32,768 GB of memory. The LogHub record stores an alert-category field in the first column: "-" denotes a non-alert message and any other category denotes an alert. The widely used raw stream contains approximately 4.7–5 million messages; the literature reports 143 alert categories, while the exact number of anomalous windows changes with the event-window definition. The labels are message-level alert indicators rather than template labels. We use the chronological stream and retain alert categories only for descriptive analysis.

Thunderbird. Thunderbird contains logs from the Thunderbird supercomputer at Sandia National Laboratories, with 9,024 processors and 27,072 GB of memory. As in BGL, the first field encodes alert status: "-" is non-alert and other values are alert categories. The retained time span determines the raw-record count; our main-paper working corpus contains approximately 10 million messages, and the source documentation reports 95 alert categories. We preserve the original temporal order and derive window labels from retained message-level alerts.

Hadoop. Hadoop contains MapReduce logs collected from a five-machine cluster with 46 CPU cores and 16 GB memory per machine. WordCount and PageRank are executed under normal conditions and under three injected deployment failures: machine shutdown, network disconnection, and disk-full. The release provides abnormal job identifiers rather than assigning labels to templates. The working corpus contains approximately 394 thousand log messages, and window labels are inherited from the provided job-level abnormality.

For all five systems, Drain uses one configuration shared by all methods. The template text and extracted parameters are the only parser outputs exposed to the models. Sentence-BERT supplies a 768-dimensional template embedding, and the character encoder represents the extracted parameter string. No anomaly-specific information is used during parsing or representation pre-computation. Empty parameter fields use a fixed sentinel string, and messages for which Drain returns no parameter span retain their complete template text. Labels are inherited from the original message, block, trace, or job annotations only after windows have been formed; no label is assigned to a template.

B.2 Window Construction and Data Splits

Messages are grouped chronologically into non-overlapping windows of $T = 20$ events. A window is positive if it contains at least one anomalous event, and negative otherwise. For the standard protocol, each dataset is split chronologically at the window level: the first 80% of windows form the training partition and the remaining 20% form the test partition. The temporal order inside every window and across both partitions is preserved; no window is shuffled across the split. The same ordered partitions, labels, and windows are given to all baselines. The stress-test subsets contain 100K consecutive windows per system and preserve the original anomaly ratio (approximately 2.7%); the final 80K windows are fixed for testing and the preceding training prefix is varied over 5K, 10K, 15K, and 20K. Cross-dataset adaptation pretrains on the source system and fine-tunes on exactly 20K labeled target windows, selected as the earliest target training prefix. All reported metrics are computed on the untouched target test partition.

The split unit is a complete window, never an individual event. Consequently, no events from a test window are used to construct a training example. Windows are generated once and their chronolog-

ical indices are stored before model training, so every baseline sees exactly the same examples. The fixed seed controls parameter initialization and minibatch composition only; it never changes temporal ordering or the train/test boundary. During training, batches contain only training windows. The test loader preserves chronological order, is never used for model selection, and is evaluated once after training. The operating point is fixed at logit threshold 0 (equivalently probability threshold 0.5), unless a baseline requires a method-specific threshold, in which case that threshold is fitted on the training partition and then frozen for testing.

For a test set with confusion-matrix counts (TP, FP, FN, TN), we compute Precision as $P = \frac{TP}{TP+FP}$, Recall as $R = \frac{TP}{TP+FN}$, and F1 as $F1 = \frac{2PR}{P+R}$. All metrics are computed from the pooled window-level predictions over the entire test set rather than by averaging per-batch scores, which avoids assigning disproportionate weight to batches containing a small number of anomalous windows and is particularly important for highly imbalanced log datasets.

B.3 Implementation Details of LLM-Based Methods

GPT-4o and DeepSeek-Flash are evaluated using a fixed two-stage prompting protocol that mirrors the preprocessing and decision pipeline.

Table 7: Two-stage prompting protocol for the LLM-based baselines.

LLM-Based Log Analysis	

Input	`2026-06-02 10:30:00 DEBUG: Checking server availability.`
	`2026-06-02 10:30:15 ERROR: NetworkException - Unable to establish connection.`
	`2026-06-02 10:31:00 INFO: Retrying connection to server.`

Hop 1: Parsing	Extract event templates and variable fields from the ordered raw log sequence while preserving every event and its original position.
Hop 2: Detection	Use the extracted templates, variables, and sequential context to determine whether the complete log window contains anomalous behavior.
Output	`Yes` (anomalous) or `No` (normal)

The first hop performs label-free structural parsing of each log window. Given the raw chronological events, it identifies recurring event patterns and decomposes each event into a template component and variable fields. The resulting LLM-induced representation is passed to the second hop together with the original event sequence, enabling the detector to jointly access lexical content, temporal ordering, and structural regularities. No externally generated Drain templates, character encodings, sentence embeddings, or other engineered features are provided to either hop. Both hops use fixed task-specific prompts with a temperature of 0.25 and deterministic prompt formatting across all datasets.

For each test window, the first hop is executed independently and its output is used only as intermediate context for the corresponding second-hop prediction. The second prompt contains the generated structural description followed by the original ordered events, with explicit event separators to preserve window boundaries and temporal order. Timestamps are retained when available, while filenames, labels, split indicators, and other dataset-specific metadata are excluded. The second hop is constrained to output a single binary decision: `Yes` denotes an anomalous window and `No` denotes a normal window. No in-context demonstrations, test labels, cross-window context, or external retrieval are used.

Outputs are normalized by trimming leading and trailing whitespace and applying case-insensitive matching. Only exact matches to the two permitted labels are accepted; malformed or additional-text responses are counted as incorrect predictions rather than heuristically mapped to either class. The predictions are evaluated directly at the window level using Precision, Recall, and F1, with identical test windows and label definitions across all LLMs. No confidence thresholding, calibration, voting, or external post-processing is applied. This setup isolates the zero-shot detection capability of the LLMs while ensuring that the comparison does not benefit from dataset-specific parsers or separately engineered feature representations.

B.4 LLM-BASED NOISE GENERATION PROCEDURE

Noise is injected into the log messages before embedding, while the original window boundaries and window-level binary labels are retained. For a target corruption rate $\alpha \in 0.01, 0.02, 0.05$, the generator selects approximately αN messages and applies one of four corruption operators: (i) template-level replacement or deletion of a non-critical token, (ii) parameter-type-preserving substitution, (iii) local token insertion, deletion, or permutation, and (iv) value perturbation for identifiers, paths, addresses, and numeric fields. The generator receives the message, its parsed template, parameter types, and neighboring messages, and is required to preserve the timestamp format, field boundaries, and number of log records. A validator rejects edits that change the record count, break the parser format, or violate the semantic constraints required to preserve the original window label. The corrupted messages are then assigned to the same train/test partitions as their corresponding clean messages. For each α, the same fixed corrupted partitions are used by all methods, while the clean corpus is never mixed into the noisy test partition. This protocol isolates robustness to observation corruption rather than label noise or changes in the underlying system behavior.

The corruption rate α is defined over messages rather than tokens or windows. Sampling is stratified by the window-level binary label so that corruption does not systematically change the anomaly prevalence. When an operator cannot be safely applied to a selected message, the message is replaced by another message from the same stratum. Each generated edit is recorded together with its operator type, source message, and validator status, allowing the aggregate corruption rate and operator frequencies to be audited independently of model predictions. For each corruption rate, we generate one fixed corrupted corpus and reuse the resulting train/test partitions for every method, thereby avoiding method-dependent perturbations. The original labels are retained because the experimental objective is to measure invariance to observation corruption; edits that alter the underlying fault semantics are rejected rather than relabeled.

C ADDITIONAL EXPERIMENTS

We conduct our analysis on the 100K-instance stress-test subsets introduced in the previous section.

C.1 BACKBONE COMPARISONS

Table 8 evaluates whether the proposed behavioral-dynamics formulation can be replaced by generic sequence backbones. Across HDFS, BGL, and Hadoop, which exhibit different temporal and contextual characteristics, PulseLog consistently achieves the highest F1 scores of 99.51%, 96.15%, and 99.77%, respectively. This consistent improvement suggests that the proposed context-conditioned transition mechanism is more effective at capturing behavioral deviations than generic sequence modeling.

Table 8: Comparison of different backbones.

Backbone	HDFS			BGL			Hadoop			Params	FLOPs
	P	R	F1	P	R	F1	P	R	F1		
GRU (Cho et al., 2014)	0.9552	0.9381	0.9466	0.7865	0.9420	0.8573	0.9651	0.9742	0.9696	0.86M	28.70M
Transformer (Vaswani et al., 2017)	0.9835	0.9768	0.9801	0.9078	**0.9736**	0.9395	0.9901	0.9940	0.9920	2.05M	72.80M
TCN (Bai et al., 2018)	0.9638	0.9475	0.9556	0.8016	0.9502	0.8696	0.9725	0.9810	0.9767	0.93M	25.90M
S4D (Smith et al., 2022)	0.9781	0.9635	0.9707	0.8424	0.9732	0.9031	0.9859	0.9912	0.9885	1.02M	32.60M
Mamba (Gu & Dao, 2023)	0.9812	0.9729	0.9770	0.8854	0.9658	0.9239	0.9912	0.9934	0.9923	1.18M	40.20M
PulseLog	**0.9954**	**0.9948**	**0.9951**	**1.0000**	0.9259	**0.9615**	**0.9987**	**0.9968**	**0.9977**	1.20M	41.43M

The component-level results further reveal how this advantage arises. On HDFS and Hadoop, PulseLog achieves the highest precision and recall simultaneously, indicating that its selective state evolution can better distinguish genuine failure transitions from rare but contextually compatible events. On BGL, PulseLog achieves perfect precision and the highest F1, although Transformer obtains a higher recall (97.36% vs. 92.59%). This trade-off suggests that the context-gated dynamics can effectively suppress locally unusual events that remain compatible with the current behavioral regime, while retaining events that provide sufficient evidence of a genuine deviation.

The improvement is not simply a consequence of model capacity. PulseLog uses only 1.20M parameters and 41.43M FLOPs, comparable to Mamba (1.18M parameters and 40.20M FLOPs) and substantially smaller than Transformer-small (2.05M parameters and 72.80M FLOPs). In contrast,

the lower-cost GRU and TCN do not achieve comparable F1 scores. These results indicate that the performance gain primarily comes from adaptive state retention and context-dependent evidence injection, rather than from increased generic sequence-modeling capacity.

C.2 Fusion Mechanism Comparisons

Table 9 evaluates the effect of separating persistent and transient evidence. The persistent-only variant performs substantially worse across all three datasets, achieving F1 scores of 97.85%, 91.92%, and 98.74% on HDFS, BGL, and Hadoop, respectively. The degradation is most pronounced on BGL, where short-lived alerts can be diluted by a representation dominated by the persistent operating regime. A similar effect is observed on Hadoop, where failures may manifest as short behavioral transitions within longer application traces.

Table 9: Comparison of different fusion strategies.

Fusion	HDFS			BGL			Hadoop			Params	FLOPs
	P	R	F1	P	R	F1	P	R	F1		
Persistent only	0.9847	0.9723	0.9785	0.9620	0.8800	0.9192	0.9908	0.9841	0.9874	1.17M	39.60M
Concat + MLP	0.9886	0.9804	0.9845	0.9740	0.9000	0.9355	0.9937	0.9892	0.9914	1.19M	40.20M
Feature-wise gate	0.9908	0.9852	0.9880	0.9638	0.9271	0.9451	0.9953	0.9918	0.9935	1.19M	40.50M
FiLM fusion (Perez et al., 2018)	0.9917	0.9871	0.9894	0.9870	0.9140	0.9491	0.9961	0.9934	0.9947	1.24M	43.10M
Cross-attention (Tan & Bansal, 2019)	0.9936	0.9924	0.9930	0.9858	**0.9317**	0.9580	0.9976	0.9963	0.9970	1.55M	56.80M
PulseLog router	**0.9954**	**0.9948**	**0.9951**	**1.0000**	0.9259	**0.9615**	**0.9987**	**0.9968**	**0.9977**	1.20M	41.43M

Introducing the transient branch improves simple concatenation to 98.45%, 93.55%, and 99.14% F1, while the full multi-rate router further increases these scores to 99.51%, 96.15%, and 99.77%. The additional improvement over simple concatenation suggests that the benefit comes not only from incorporating transient evidence, but also from dynamically weighting evidence at different temporal scales. The router can retain persistent evidence when the execution remains stable while emphasizing transient changes when they become informative under the current context. Cross-attention achieves competitive performance on several individual metrics, but requires 1.55M parameters and 56.80M FLOPs and remains below the router in F1 on all three datasets. Overall, the results show that multi-rate evidence separation combined with lightweight context-dependent routing provides a more effective trade-off between detection performance and computational cost.

C.3 Trajectory readout Strategy Comparisons

Table 10 evaluates the final design choice: anomaly decisions should account for behavioral transitions rather than relying solely on a pooled state representation. Across HDFS, BGL, and Hadoop, the PulseLog Deviation Probe achieves F1 scores of 99.51%, 96.15%, and 99.77%, respectively, improving over the strongest alternative by 0.18, 0.53, and 0.18 percentage points.

Table 10: Comparison of different trajectory readout strategies.

Trajectory Readout	HDFS			BGL			Hadoop			Params	FLOPs
	P	R	F1	P	R	F1	P	R	F1		
h_t + linear classifier	0.9823	0.9726	0.9774	0.9527	0.9013	0.9263	0.9892	0.9864	0.9878	1.16M	39.10M
mean pooling + MLP	0.9884	0.9807	0.9845	0.9683	0.9147	0.9407	0.9921	0.9886	0.9903	1.18M	39.80M
max pooling + MLP	0.9867	0.9782	0.9824	0.9608	0.9243	0.9422	0.9913	0.9881	0.9897	1.18M	39.80M
mean + max pooling	0.9908	0.9867	0.9887	0.9809	**0.9328**	0.9562	0.9952	0.9926	0.9939	1.19M	40.20M
Δh_t pooling	0.9932	0.9887	0.9909	0.9884	0.9148	0.9502	0.9961	0.9937	0.9949	1.19M	40.60M
$[h_t, \Delta h_t]$ pooling	0.9943	0.9924	0.9933	0.9927	0.9216	0.9558	0.9972	0.9947	0.9959	1.20M	41.10M
PulseLog Deviation Probe	**0.9954**	**0.9948**	**0.9951**	**1.0000**	0.9259	**0.9615**	**0.9987**	**0.9968**	**0.9977**	1.20M	41.43M

The comparison among readout strategies further clarifies the role of transition information. Pooling only h_t captures the current behavioral state but discards explicit change information, whereas pooling only Δh_t captures local changes without the persistent regime needed to interpret whether those changes are expected. Concatenating h_t and Δh_t recovers both signals, but remains insensitive to their temporal interaction. In contrast, the Deviation Probe explicitly evaluates the change relative to the current behavioral context. This distinction is particularly beneficial on BGL, where Mean–max pooling achieves slightly higher recall (93.28%), but the Deviation Probe maintains perfect precision and obtains the higher F1 score. Overall, the results suggest that anomaly detection benefits from modeling whether a behavioral change is compatible with its surrounding context, rather than treating every salient transition as anomalous.

D VISUALIZATION ANALYTICS

This analysis is conducted on the original datasets.

D.1 NORMAL AND ANOMALOUS FEATURES

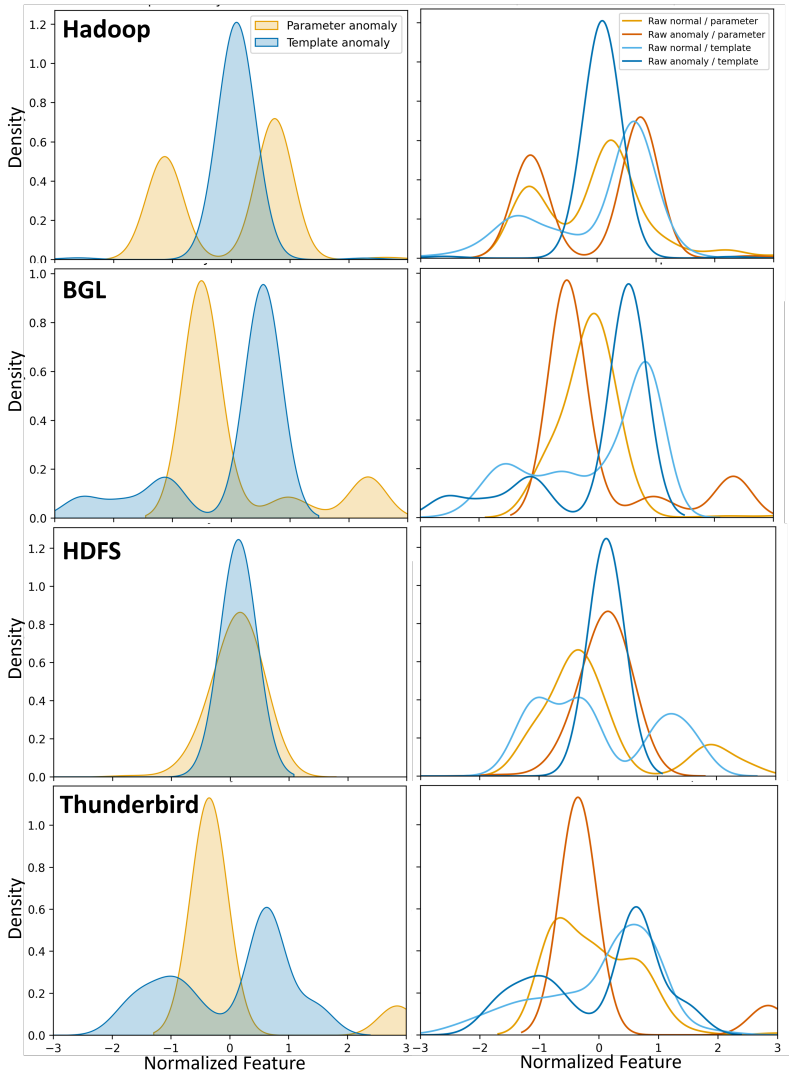

Figure 3: Distributional analysis of template- and parameter-level evidence across Hadoop, BGL, HDFS, and Thunderbird. The left column compares raw parameter and template representations for normal and anomalous events, while the right column shows the corresponding class-conditional density curves. The figure illustrates how the reliability and separability of local evidence vary across systems.

Figure 3 provides input-level evidence motivating PulseLog's multi-rate, context-conditioned formulation. Across the four systems, template semantics, runtime parameters, and anomaly labels exhibit distinct distributional structures, suggesting that the reliability of individual event features is inherently context dependent. Moreover, the evidence exhibits different temporal characteristics: persistent patterns characterize stable operating regimes, whereas transient patterns capture short-lived deviations. A detector that applies a uniform decision rule or prematurely couples these signals may therefore obscure their distinct discriminative structures.

Hadoop. Template and parameter distributions exhibit relatively clear marginal separation, yet their class-conditional densities remain multi-modal. This behavior is consistent with the heterogeneous WordCount and PageRank workloads and failures spanning machine, network, and disk levels. In particular, the same parameter value may be benign in one execution context but anomalous in another. Moreover, template and parameter densities peak at different locations, indicating that neither representation alone provides a sufficient decision signal and motivating their explicit fusion under runtime context.

BGL. The normal and anomalous distributions contain sharp, localized modes, reflecting a highly regular stream in which anomalies appear as sparse deviations from stable operating regimes. Importantly, the distributions suggest complementary separability between persistent and transient evidence: persistent patterns characterize the dominant operating regime, whereas transient patterns capture localized deviations from that regime. Directly coupling these signals into a single representation can obscure this distinction, allowing strong persistent patterns to dominate short-lived deviations or transient variations to perturb the estimation of stable background behavior. PulseLog therefore extracts the two evidence types through separate multi-rate branches and fuses them only after rate-specific dynamics have been established. The persistent branch captures regime-level evidence, while the transient branch preserves localized changes that may otherwise be suppressed by temporal aggregation.

HDFS. Normal and anomalous parameter distributions exhibit substantial overlap, while template evidence is more concentrated. Many events are shared between normal and failed block traces, making individual event features weak discriminators. The distinction instead emerges from event ordering and accumulation over the trace. This structure motivates modeling anomalies as deviations in latent state transitions rather than as pointwise feature outliers. At the same time, separating persistent and transient evidence allows stable block-level behavior to be distinguished from short-lived deviations, preventing transient variations from being absorbed into the background representation.

Thunderbird. Template and parameter densities are broad and multi-modal, with pronounced tails characteristic of a diverse, long-running system stream. Such distributional heterogeneity makes fixed thresholds on individual fields brittle. Persistent aggregation can capture dominant operating regimes, whereas the transient branch retains short-lived departures from these regimes. Keeping these signals separate before fusion preserves their respective discriminative structures, while context-conditioned integration determines whether a transient deviation is compatible with the current behavioral state. The Deviation Probe then evaluates whether the resulting deviation constitutes an incompatible ordered transition.

Taken together, these distributions show that event evidence is heterogeneous across both systems and temporal rates. Template features capture coarse regime structure, while runtime parameters reflect execution-specific variation. Persistent and transient dynamics provide complementary discriminative signals.

These evidence types should not be fused indiscriminately. Early fusion can mix stable background patterns with short-lived deviations and reduce the separability of individual signals. PulseLog therefore adopts a separation-before-fusion principle. Persistent and transient evidence are first modeled by dedicated multi-rate branches to preserve their distinct temporal dynamics. They are then integrated under runtime context. Finally, their interaction is evaluated at the trajectory level. This design helps distinguish stable operating behavior from transient deviations and detects anomalies as incompatible state transitions rather than isolated event outliers.

D.2 Raw and Behavioral Features

Figure 4 compares the raw input representations with the behavioral features learned by PulseLog. In the raw-feature panels, normal and anomalous windows exhibit substantial overlap, with scattered outliers across the feature space. This reflects a key limitation of event-level representations: individual templates or runtime parameters are not uniformly informative of anomalous behavior. Rare events may occur during normal execution, while anomalies may instead manifest as changes in event ordering or system dynamics without producing clearly separable raw features.

In contrast, the behavioral-feature panels exhibit more structured class organization. Normal windows tend to form denser local regions, while anomalous windows are more frequently concentrated

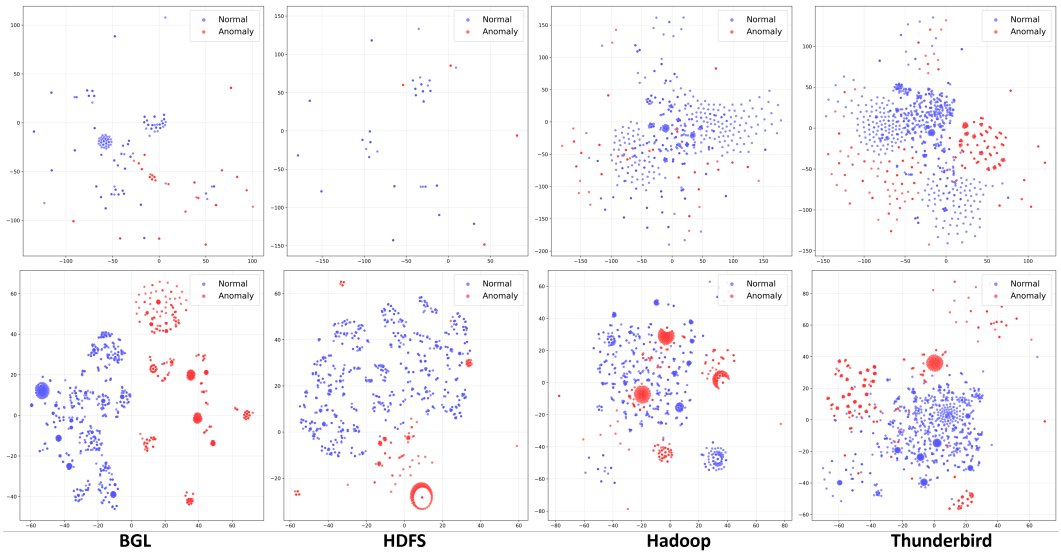

Figure 4: t-SNE visualization of raw and PulseLog behavioral features for normal and anomalous windows. Blue and red indicate normal and anomalous labels, respectively. The upper row shows raw features and the lower row shows the corresponding behavioral representations for four datasets.

in distinct regions or clusters. This structure is consistent with PulseLog's representation design. Multi-rate aggregation captures both persistent operating regimes and transient deviations, while context-conditioned dynamics modulates state updates according to the surrounding execution context. The Deviation Probe then summarizes the resulting ordered state transitions for window-level prediction. Consequently, the learned representation reflects behavioral dynamics rather than isolated event similarity.

The separation remains imperfect across datasets, and the t-SNE projections should not be interpreted as quantitative evidence of classification performance. Instead, they provide qualitative evidence that PulseLog transforms raw log observations into more structured representations of context-dependent behavior. This supports our central modeling perspective: log anomalies are better characterized as deviations in context-specific state transitions than as isolated outliers in the raw feature space.

E Sensitivity Analysis

We conduct our analysis on the 100K-instance stress-test subsets introduced in the previous section.

E.1 Sensitivity to Window Length

Figure 5 shows that window length has a non-monotonic effect on detection performance while directly increasing sequential computation. Across all three datasets, $T = 20$ achieves the highest F1, reaching 0.9951 on HDFS, 0.9615 on BGL, and 0.9977 on Hadoop. Reducing the window to $T = 10$ decreases F1 by 0.84, 2.73, and 0.69 percentage points, respectively. The larger degradation on BGL is consistent with its sparse alert events: shorter windows provide less surrounding context for distinguishing genuine alert-related deviations from isolated fluctuations. HDFS and Hadoop similarly benefit from additional context for characterizing block-level behavior and workload-dependent failures.

Further increasing the window length does not yield additional gains. At $T = 30$, F1 decreases to 0.9934, 0.9578, and 0.9963, and drops further to 0.9886, 0.9493, and 0.9931 at $T = 40$ on HDFS, BGL, and Hadoop, respectively. Longer windows can incorporate increasingly heterogeneous events, diluting localized state transitions and weakening the contribution of transient evi-

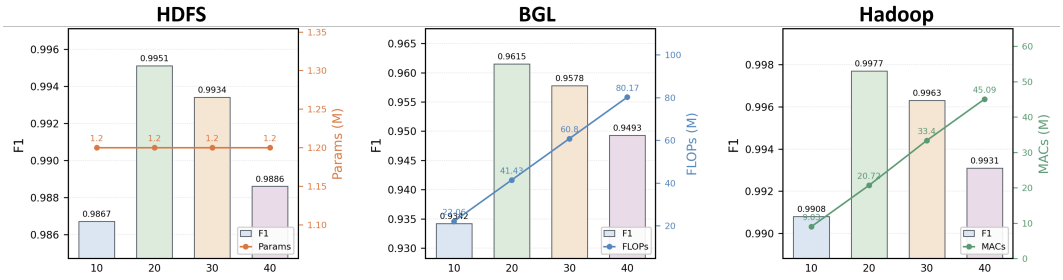

Figure 5: Effect of window length on detection performance and computational cost. Bars report F1 on HDFS, BGL, and Hadoop. The overlaid lines report parameter count, FLOPs, and MACs.

dence. These results suggest that an effective context window should be sufficiently long to establish the underlying operating regime, yet compact enough to preserve localized behavioral deviations.

The computational cost exhibits the expected dependence on sequence length. The parameter count remains fixed at 1.20M because varying T does not modify the model architecture, whereas FLOPs increase from 22.06M to 80.17M and MACs from 9.03M to 45.09M as T grows from 10 to 40. Thus, $T = 20$ provides the best empirical detection performance across all three datasets while avoiding the substantially higher computational cost of longer contexts, and is therefore adopted as the default window length.

E.2 Sensitivity to Hidden Dimension and Model Depth

We first vary the hidden dimension while fixing the dynamics depth at two layers and the window length at $T = 20$.

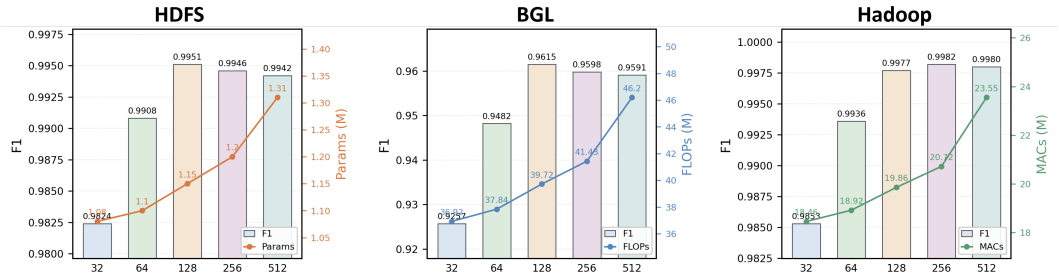

Figure 6: Sensitivity to hidden dimension. Bars show F1 on HDFS, BGL, and Hadoop; lines show parameter count, FLOPs, and MACs, respectively.

Figure 6 evaluates the effect of the hidden dimension on detection performance. Increasing the dimension from 32 to 128 consistently improves F1 across all three datasets, from 0.9824 to 0.9951 on HDFS, from 0.9257 to 0.9615 on BGL, and from 0.9853 to 0.9977 on Hadoop. This improvement suggests that a wider latent state provides greater capacity to jointly encode persistent execution regimes and localized behavioral deviations. The gain is particularly pronounced on BGL, where sparse alert events must be distinguished from a highly regular background.

Beyond 128 dimensions, the performance gains largely saturate. On HDFS, F1 changes only slightly from 0.9951 at 128 to 0.9946 and 0.9942 at 256 and 512, respectively. BGL shows a similar trend, decreasing from 0.9615 to 0.9598 and 0.9591. Hadoop is the only dataset that benefits marginally from further widening, reaching its highest F1 of 0.9982 at 256 and remaining nearly unchanged at 0.9980 at 512. Overall, these results indicate that increasing latent-state capacity is beneficial when the representation is narrow, but provides limited additional discriminative capacity once a sufficient state dimension is reached.

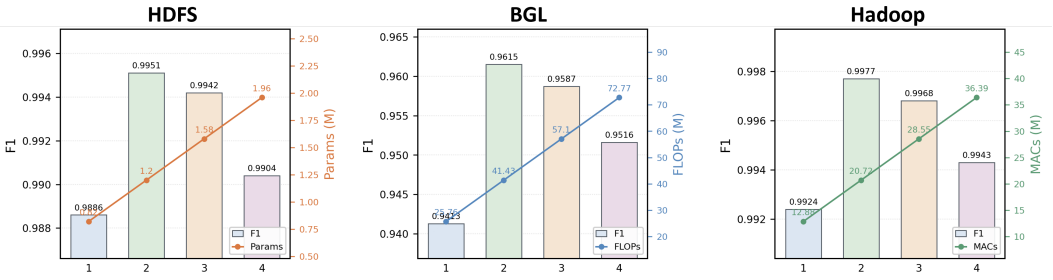

Figure 7: Sensitivity to dynamics depth. Bars show F1 on HDFS, BGL, and Hadoop; lines show parameter count, FLOPs, and MACs, respectively.

The computational cost increases more gradually than the initial accuracy gains. As the hidden dimension grows from 32 to 512, the parameter count increases from 1.08M to 1.31M, while FLOPs and MACs increase from 36.92M to 46.20M and from 18.46M to 23.55M, respectively. The 128-dimensional setting therefore provides a compact representation with strong performance, achieving the highest F1 on HDFS and BGL and near-peak performance on Hadoop at 1.15M parameters, 39.72M FLOPs, and 19.86M MACs. We nevertheless retain the 256-dimensional state as the default configuration for the main model, as it yields the highest observed Hadoop F1 with only a moderate computational overhead.

We next vary the number of stacked dynamics layers while fixing the hidden dimension at 256 and the window length at $T = 20$.

Figure 7 evaluates the effect of the number of dynamics layers on detection performance. Two layers consistently achieve the strongest results across all three datasets, with F1 scores of 0.9951 on HDFS, 0.9615 on BGL, and 0.9977 on Hadoop. A single layer performs worse, particularly on BGL, where F1 drops to 0.9413. This suggests that a single transition block provides insufficient capacity to refine the context-conditioned behavioral trajectory and relate local alert events to their surrounding operating regime.

Increasing the depth beyond two layers provides no additional benefit. With four layers, F1 decreases to 0.9904, 0.9516, and 0.9943 on HDFS, BGL, and Hadoop, respectively. Under the fixed window length and training protocol, additional transition blocks therefore appear to provide limited useful abstraction and may instead introduce redundant transformations that weaken the discriminative structure of the learned trajectory.

Depth also introduces a substantially larger computational overhead than changes in hidden dimension, since each additional layer adds a complete transition block. From one to four layers, the parameter count increases from 0.82M to 1.96M, while FLOPs and MACs increase from 25.76M to 72.77M and from 12.88M to 36.39M, respectively. The two-layer configuration thus provides the strongest empirical performance with substantially lower computational cost than deeper alternatives. We consequently use two dynamics layers as the default, providing sufficient iterative state refinement without introducing unnecessary depth.

E.3 Sensitivity to the Multi-Rate Scale Ratio

Figure 8 evaluates the effect of the temporal-scale ratio between the persistent and transient branches. The results show that anomaly detection benefits from distinct temporal supports, but that excessive separation is detrimental. When the ratio is 1, the two branches operate at the same effective scale, eliminating the intended multi-rate distinction. F1 is consequently lowest across all three datasets, at 0.9816 on HDFS, 0.9184 on BGL, and 0.9872 on Hadoop. Increasing the ratio to 1.5 improves F1 to 0.9903, 0.9467, and 0.9941, respectively, while a ratio of 2 achieves the highest scores of 0.9951, 0.9615, and 0.9977. This consistent improvement supports the use of complementary temporal supports to capture both persistent operating regimes and localized deviations.

The benefit decreases when the scale gap becomes too large. At ratios of 3 and 4, F1 falls from its peak on each dataset; at 4, it reaches 0.9879 on HDFS, 0.9413 on BGL, and 0.9926 on Hadoop.

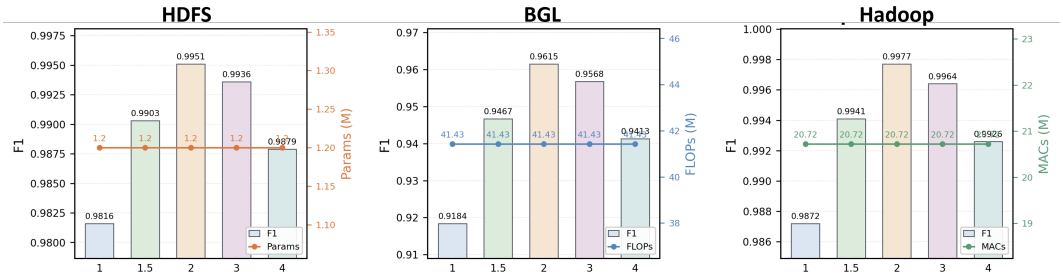

Figure 8: Sensitivity to the ratio between persistent and transient temporal scales. Bars report F1 on HDFS, BGL, and Hadoop; lines report parameter count, FLOPs, and MACs, respectively.

Excessive separation can make the persistent branch overly smooth and the transient branch too narrow, weakening their ability to provide complementary evidence for context-conditioned transitions. BGL shows the largest absolute gain from ratio 1 to 2 (4.31 percentage points), consistent with its sparse alerts against a stable background: it particularly benefits from distinguishing persistent regime evidence from short-lived alert deviations.

The parameter count (1.20M), FLOPs (41.43M), and MACs (20.72M) remain constant across ratios because this hyperparameter changes the temporal supports, not the model architecture. The performance variation therefore isolates the effect of multi-rate separation itself. A ratio of 2 provides the best measured balance between preserving persistent behavior and retaining transient changes, supporting its use as the default scale ratio.

F Detailed Results for Main-Text Tables

F.1 Detailed Results for Table 1

Table 1 summarizes the results across five datasets. We additionally report the per-dataset mean and standard deviation of precision, recall, and F1.

Table 11: Detailed baseline results on HDFS_v1 (mean ± standard deviation).

Method	P	R	F1
DeepLog	0.9840 ± 0.0012	0.7865 ± 0.0028	0.8743 ± 0.0021
GAT	0.9836 ± 0.0010	0.7779 ± 0.0032	0.8687 ± 0.0024
KNN	0.8416 ± 0.0038	0.7432 ± 0.0041	0.7892 ± 0.0036
LogAnomaly	0.9831 ± 0.0011	0.7781 ± 0.0030	0.8687 ± 0.0023
LogDLR	0.8913 ± 0.0027	0.8011 ± 0.0034	0.8438 ± 0.0028
LogFormer	0.9699 ± 0.0017	0.6555 ± 0.0042	0.7823 ± 0.0034
Lograph	0.9830 ± 0.0012	0.7865 ± 0.0027	0.8739 ± 0.0020
NeuralLog	0.8679 ± 0.0031	0.7756 ± 0.0035	0.8192 ± 0.0029
PCA	0.3080 ± 0.0062	0.6865 ± 0.0051	0.4253 ± 0.0057
PLELog	0.9797 ± 0.0013	0.7822 ± 0.0029	0.8699 ± 0.0022
RobustLog	0.9218 ± 0.0023	0.7592 ± 0.0038	0.8326 ± 0.0029
DeepSeek-Flash	0.6818 ± 0.0052	1.0000 ± 0.0000	0.8108 ± 0.0041
GPT-4o	0.7926 ± 0.0041	0.9235 ± 0.0032	0.8530 ± 0.0035
LogSD	0.9852 ± 0.0010	0.7828 ± 0.0029	0.8724 ± 0.0022
MIDLog	0.7785 ± 0.0039	0.9841 ± 0.0010	0.8693 ± 0.0026
PulseLog	0.8457 ± 0.0028	0.9637 ± 0.0018	0.9009 ± 0.0021

On HDFS_v1, as shown in Table 11, PulseLog achieves standard deviations of 0.0028, 0.0018, and 0.0021 for precision, recall, and F1, respectively, indicating low run-to-run variability. Its F1 mean is the highest, with a 0.0266 margin over the next method. This gap is descriptive rather than a statistical significance claim, as significance testing would require seed-level scores and paired comparisons. DeepSeek-Flash exhibits zero recall variance at the reported ceiling, while showing

substantially lower precision and F1, indicating that recall stability alone does not imply uniformly stable performance.

Table 12: Detailed baseline results on HDFS_v3 (mean ± standard deviation).

Method	P	R	F1
DeepLog	0.9985 ± 0.0003	0.8820 ± 0.0021	0.9366 ± 0.0014
GAT	0.9890 ± 0.0011	0.8746 ± 0.0026	0.9283 ± 0.0018
KNN	0.9138 ± 0.0029	0.8496 ± 0.0037	0.8805 ± 0.0031
LogAnomaly	0.9969 ± 0.0004	0.8766 ± 0.0024	0.9329 ± 0.0016
LogDLR	0.9983 ± 0.0003	0.8818 ± 0.0020	0.9364 ± 0.0015
LogFormer	0.7612 ± 0.0038	0.6724 ± 0.0045	0.7140 ± 0.0039
Lograph	0.9975 ± 0.0004	0.8806 ± 0.0021	0.9354 ± 0.0015
NeuralLog	0.9960 ± 0.0006	0.7871 ± 0.0032	0.8793 ± 0.0025
PCA	0.6768 ± 0.0048	0.8637 ± 0.0039	0.7589 ± 0.0042
PLELog	0.9979 ± 0.0003	0.8778 ± 0.0022	0.9340 ± 0.0016
RobustLog	0.9962 ± 0.0005	0.8761 ± 0.0023	0.9323 ± 0.0017
DeepSeek-Flash	0.7136 ± 0.0047	0.7948 ± 0.0042	0.7510 ± 0.0044
GPT-4o	0.7618 ± 0.0044	0.8427 ± 0.0038	0.7992 ± 0.0039
LogSD	0.9975 ± 0.0004	0.8816 ± 0.0020	0.9360 ± 0.0015
MIDLog	0.8815 ± 0.0028	0.9977 ± 0.0004	0.9360 ± 0.0014
PulseLog	0.9991 ± 0.0002	0.8818 ± 0.0017	0.9368 ± 0.0014

On HDFS_v3, as shown in Table 12, PulseLog shows low dispersion in precision (SD 0.0002) and modest dispersion in recall and F1 (0.0017 and 0.0014, respectively). Its mean F1 exceeds DeepLog by only 0.0002, while both have an F1 SD of 0.0014. Thus, the reported summaries do not resolve the difference between the two methods; seed-level paired results would be required to assess whether the observed gap is reliable.

Table 13: Detailed baseline results on BGL (mean ± standard deviation).

Method	P	R	F1
DeepLog	0.9100 ± 0.0027	0.9764 ± 0.0012	0.9420 ± 0.0018
GAT	0.9252 ± 0.0024	0.9720 ± 0.0015	0.9481 ± 0.0017
KNN	0.9516 ± 0.0021	0.9098 ± 0.0034	0.9302 ± 0.0027
LogAnomaly	0.9819 ± 0.0015	0.9744 ± 0.0013	0.9781 ± 0.0012
LogDLR	0.9953 ± 0.0007	0.9770 ± 0.0011	0.9861 ± 0.0009
LogFormer	0.9968 ± 0.0005	0.7267 ± 0.0036	0.8406 ± 0.0028
Lograph	0.9460 ± 0.0022	0.9740 ± 0.0014	0.9598 ± 0.0018
NeuralLog	0.9962 ± 0.0006	0.9680 ± 0.0017	0.9819 ± 0.0011
PCA	0.1238 ± 0.0071	0.9864 ± 0.0011	0.2200 ± 0.0065
PLELog	0.9760 ± 0.0016	0.9740 ± 0.0014	0.9750 ± 0.0013
RobustLog	0.9882 ± 0.0010	0.9740 ± 0.0015	0.9811 ± 0.0011
DeepSeek-Flash	0.6673 ± 0.0056	0.5958 ± 0.0051	0.6297 ± 0.0053
GPT-4o	0.7786 ± 0.0040	0.7217 ± 0.0045	0.7481 ± 0.0042
LogSD	0.9932 ± 0.0008	0.9787 ± 0.0011	0.9859 ± 0.0009
MIDLog	0.9747 ± 0.0015	0.9225 ± 0.0028	0.9479 ± 0.0020
PulseLog	0.9997 ± 0.0002	0.9744 ± 0.0012	0.9869 ± 0.0008

On BGL, as shown in Table 13, PulseLog exhibits low run-to-run dispersion, with SDs of 0.0002, 0.0012, and 0.0008 for precision, recall, and F1, respectively. Its mean F1 exceeds LogDLR by 0.0008, while their F1 SDs are 0.0008 and 0.0009, respectively; this small gap does not establish a statistically significant difference without per-seed paired results. PCA achieves high recall but substantially lower precision and larger F1 dispersion (SD 0.0065), indicating less consistent overall performance.

On Hadoop, as shown in Table 14, PulseLog shows SDs of 0.0024, 0.0008, and 0.0015 for precision, recall, and F1, respectively. Its mean F1 exceeds PLELog by 0.0044, with corresponding F1 SDs of 0.0015 and 0.0017; this margin is descriptive rather than a formal significance result. KNN achieves higher precision but lower recall and F1, while methods with perfect recall obtain lower

Table 14: Detailed baseline results on Hadoop (mean ± standard deviation).

Method	P	R	F1
DeepLog	0.8527 ± 0.0031	0.9935 ± 0.0008	0.9177 ± 0.0019
GAT	0.8579 ± 0.0027	0.9968 ± 0.0005	0.9221 ± 0.0018
KNN	0.9427 ± 0.0024	0.8618 ± 0.0038	0.9004 ± 0.0030
LogAnomaly	0.8560 ± 0.0028	0.9935 ± 0.0008	0.9197 ± 0.0017
LogDLR	0.8634 ± 0.0025	0.9864 ± 0.0010	0.9208 ± 0.0018
LogFormer	0.8270 ± 0.0031	1.0000 ± 0.0000	0.9053 ± 0.0026
Lograph	0.8547 ± 0.0029	0.9909 ± 0.0009	0.9178 ± 0.0018
NeuralLog	0.8270 ± 0.0030	1.0000 ± 0.0000	0.9053 ± 0.0024
PCA	0.8520 ± 0.0042	0.7306 ± 0.0053	0.7866 ± 0.0047
PLELog	0.8571 ± 0.0027	0.9981 ± 0.0004	0.9222 ± 0.0017
RobustLog	0.8570 ± 0.0028	0.9935 ± 0.0008	0.9202 ± 0.0018
DeepSeek-Flash	0.7483 ± 0.0048	0.6759 ± 0.0050	0.7099 ± 0.0046
GPT-4o	0.7924 ± 0.0040	0.7486 ± 0.0043	0.7699 ± 0.0041
LogSD	0.8523 ± 0.0029	0.9942 ± 0.0007	0.9178 ± 0.0018
MIDLog	0.8535 ± 0.0029	0.9922 ± 0.0008	0.9176 ± 0.0018
PulseLog	0.8681 ± 0.0024	0.9935 ± 0.0008	0.9266 ± 0.0015

F1, highlighting the need to assess stability jointly across metrics rather than at a single operating point.

Table 15: Detailed baseline results on Thunderbird (mean ± standard deviation).

Method	P	R	F1
DeepLog	0.9999 ± 0.0001	1.0000 ± 0.0000	1.0000 ± 0.0000
GAT	0.9999 ± 0.0001	1.0000 ± 0.0000	1.0000 ± 0.0000
KNN	0.9942 ± 0.0007	0.9785 ± 0.0012	0.9863 ± 0.0009
LogAnomaly	0.9999 ± 0.0001	1.0000 ± 0.0000	1.0000 ± 0.0000
LogDLR	0.9999 ± 0.0001	0.9999 ± 0.0001	0.9999 ± 0.0001
LogFormer	0.9986 ± 0.0003	0.9991 ± 0.0002	0.9988 ± 0.0002
Lograph	0.9999 ± 0.0001	0.9999 ± 0.0001	0.9999 ± 0.0001
NeuralLog	0.9998 ± 0.0001	0.9998 ± 0.0001	0.9998 ± 0.0001
PCA	0.4664 ± 0.0058	0.7173 ± 0.0046	0.5652 ± 0.0051
PLELog	0.9999 ± 0.0001	1.0000 ± 0.0000	1.0000 ± 0.0000
RobustLog	0.9998 ± 0.0001	0.9999 ± 0.0001	0.9999 ± 0.0001
DeepSeek-Flash	0.9068 ± 0.0028	0.8607 ± 0.0031	0.8821 ± 0.0027
GPT-4o	0.8963 ± 0.0029	0.8457 ± 0.0032	0.8703 ± 0.0028
LogSD	0.9999 ± 0.0001	0.9999 ± 0.0001	0.9999 ± 0.0001
MIDLog	0.9997 ± 0.0002	0.9995 ± 0.0002	0.9996 ± 0.0002
PulseLog	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000

On Thunderbird, as shown in Table 15, the results reach a metric ceiling: PulseLog and several baselines report a mean of 1.0000 with SD 0.0000, indicating no observed variation at the reported precision. However, the ceiling prevents meaningful separation among these methods. Since values are rounded to four decimals, zero SD should be interpreted as no observed variation at the reporting resolution rather than exact invariance. Thus, the results primarily demonstrate stability on this benchmark rather than a distinguishable performance advantage among the saturated methods.

F.2 Detailed Results for Table 4

Table 4 reports F1 across four training budgets. We further provide the per-dataset results, grouping 5K/10K and 15K/20K budgets for each dataset.

On BGL, as shown in Table 16 and Table 17, PulseLog maintains strong and stable F1 across all four training budgets, achieving the highest F1 at each budget while maintaining competitive rather than maximal recall. At 15K, MIDLog achieves the highest precision, whereas PCA reaches perfect recall from 10K to 20K but with substantially lower F1. These results highlight the importance of balancing precision and recall under limited training data. Moreover, the consistent F1 advantage

Table 16: BGL data-efficiency results for 5K and 10K training windows

	5K			10K		
Method	P	R	F1	P	R	F1
DeepLog	0.6125 ± 0.0048	0.5847 ± 0.0062	0.5983 ± 0.0053	0.6915 ± 0.0037	0.9841 ± 0.0012	0.8122 ± 0.0029
GAT	0.6952 ± 0.0039	0.9894 ± 0.0010	0.8166 ± 0.0027	0.9683 ± 0.0018	0.9531 ± 0.0021	0.9606 ± 0.0016
KNN	0.5000 ± 0.0068	0.5344 ± 0.0075	0.5166 ± 0.0064	0.4988 ± 0.0061	0.5344 ± 0.0072	0.5160 ± 0.0060
LogAnomaly	0.7426 ± 0.0035	0.6815 ± 0.0047	0.7107 ± 0.0039	0.6694 ± 0.0042	0.8519 ± 0.0030	0.7497 ± 0.0036
LogDLR	0.9579 ± 0.0017	0.9630 ± 0.0015	0.9604 ± 0.0013	0.9042 ± 0.0025	0.9735 ± 0.0011	0.9376 ± 0.0018
LogFormer	0.7325 ± 0.0041	0.7814 ± 0.0038	0.7562 ± 0.0035	0.7648 ± 0.0036	0.8236 ± 0.0032	0.7931 ± 0.0030
Lograph	0.6815 ± 0.0046	0.6238 ± 0.0054	0.6513 ± 0.0048	0.9127 ± 0.0022	0.9683 ± 0.0014	0.9397 ± 0.0018
NeuralLog	0.6842 ± 0.0044	0.7425 ± 0.0040	0.7121 ± 0.0038	0.7216 ± 0.0037	0.7864 ± 0.0034	0.7526 ± 0.0032
PCA	0.6066 ± 0.0052	0.8810 ± 0.0028	0.7184 ± 0.0041	0.6332 ± 0.0049	1.0000 ± 0.0000	0.7754 ± 0.0036
PLELog	0.8668 ± 0.0025	0.9815 ± 0.0011	0.9206 ± 0.0019	0.7957 ± 0.0030	0.9894 ± 0.0009	0.8821 ± 0.0022
RobustLog	0.7956 ± 0.0034	0.7619 ± 0.0041	0.7784 ± 0.0036	0.7624 ± 0.0035	0.8148 ± 0.0031	0.7877 ± 0.0032
LogSD	0.8333 ± 0.0028	0.9788 ± 0.0012	0.9002 ± 0.0020	0.8383 ± 0.0027	0.9735 ± 0.0011	0.9009 ± 0.0013
MIDLog	0.5926 ± 0.0057	0.9956 ± 0.0007	0.7430 ± 0.0038	0.9815 ± 0.0010	0.7794 ± 0.0035	0.8689 ± 0.0024
PulseLog	1.0000 ± 0.0000	0.9286 ± 0.0025	0.9630 ± 0.0014	1.0000 ± 0.0000	0.9312 ± 0.0023	0.9644 ± 0.0013

Table 17: BGL results for 15K and 20K training size

	15K			20K		
Method	P	R	F1	P	R	F1
DeepLog	0.6932 ± 0.0035	0.9921 ± 0.0008	0.8161 ± 0.0025	0.6952 ± 0.0032	0.9894 ± 0.0010	0.8166 ± 0.0023
GAT	0.9786 ± 0.0013	0.9656 ± 0.0018	0.9720 ± 0.0014	0.9915 ± 0.0008	0.9233 ± 0.0026	0.9562 ± 0.0017
KNN	0.3866 ± 0.0085	0.3201 ± 0.0092	0.3502 ± 0.0080	0.5335 ± 0.0064	0.6323 ± 0.0058	0.5787 ± 0.0059
LogAnomaly	0.7484 ± 0.0031	0.9365 ± 0.0018	0.8320 ± 0.0024	0.8468 ± 0.0027	0.9032 ± 0.0021	0.8741 ± 0.0024
LogDLR	0.8676 ± 0.0029	0.9709 ± 0.0012	0.9164 ± 0.0020	0.8988 ± 0.0024	0.9630 ± 0.0014	0.9298 ± 0.0019
LogFormer	0.7935 ± 0.0031	0.8612 ± 0.0027	0.8259 ± 0.0029	0.8126 ± 0.0028	0.8795 ± 0.0024	0.8447 ± 0.0026
Lograph	0.8615 ± 0.0027	0.9709 ± 0.0013	0.9129 ± 0.0021	0.8226 ± 0.0030	0.9815 ± 0.0011	0.8951 ± 0.0020
NeuralLog	0.7489 ± 0.0036	0.8127 ± 0.0030	0.7795 ± 0.0031	0.7714 ± 0.0032	0.8386 ± 0.0028	0.8036 ± 0.0030
PCA	0.6364 ± 0.0046	1.0000 ± 0.0000	0.7778 ± 0.0034	0.6364 ± 0.0045	1.0000 ± 0.0000	0.7778 ± 0.0033
PLELog	0.8983 ± 0.0021	0.9815 ± 0.0010	0.9381 ± 0.0017	0.9185 ± 0.0018	0.9841 ± 0.0009	0.9502 ± 0.0015
RobustLog	0.7440 ± 0.0038	0.8148 ± 0.0030	0.7778 ± 0.0034	0.7951 ± 0.0031	0.8519 ± 0.0027	0.8225 ± 0.0028
LogSD	0.8440 ± 0.0026	0.9735 ± 0.0012	0.9042 ± 0.0020	0.8835 ± 0.0023	0.9233 ± 0.0020	0.9030 ± 0.0019
MIDLog	0.9894 ± 0.0008	0.7633 ± 0.0038	0.8618 ± 0.0027	0.9868 ± 0.0009	0.7803 ± 0.0034	0.8715 ± 0.0025
PulseLog	0.9661 ± 0.0020	0.9815 ± 0.0011	0.9738 ± 0.0012	1.0000 ± 0.0000	0.9259 ± 0.0026	0.9615 ± 0.0015

Table 18: HDFS results for 5K and 10K training size

	5K			10K		
Method	P	R	F1	P	R	F1
DeepLog	0.7589 ± 0.0058	0.4952 ± 0.0072	0.5993 ± 0.0061	0.9276 ± 0.0032	0.9145 ± 0.0038	0.9209 ± 0.0031
GAT	0.9085 ± 0.0034	0.9418 ± 0.0028	0.9248 ± 0.0027	0.9564 ± 0.0023	0.9782 ± 0.0017	0.9671 ± 0.0018
KNN	0.0552 ± 0.0041	0.0552 ± 0.0038	0.0552 ± 0.0039	0.5210 ± 0.0082	0.0744 ± 0.0063	0.1302 ± 0.0071
LogAnomaly	0.6861 ± 0.0052	0.9983 ± 0.0006	0.8132 ± 0.0041	0.9536 ± 0.0028	0.8702 ± 0.0037	0.9101 ± 0.0029
LogDLR	0.9867 ± 0.0011	0.9825 ± 0.0014	0.9846 ± 0.0010	0.9918 ± 0.0008	0.9857 ± 0.0012	0.9887 ± 0.0007
LogFormer	0.9148 ± 0.0032	0.8908 ± 0.0039	0.9027 ± 0.0031	0.9446 ± 0.0027	0.9190 ± 0.0034	0.9316 ± 0.0025
Lograph	0.9758 ± 0.0017	0.9365 ± 0.0028	0.9557 ± 0.0019	0.9876 ± 0.0010	0.9824 ± 0.0013	0.9850 ± 0.0009
NeuralLog	0.7826 ± 0.0048	0.9645 ± 0.0021	0.8638 ± 0.0036	0.8235 ± 0.0045	0.9726 ± 0.0018	0.8909 ± 0.0032
PCA	0.4165 ± 0.0063	0.9758 ± 0.0015	0.5850 ± 0.0048	0.4166 ± 0.0058	0.9715 ± 0.0017	0.5859 ± 0.0046
PLELog	0.9892 ± 0.0008	0.9816 ± 0.0011	0.9854 ± 0.0007	0.9932 ± 0.0006	0.9864 ± 0.0010	0.9898 ± 0.0006
RobustLog	0.9638 ± 0.0022	0.9476 ± 0.0029	0.9556 ± 0.0021	0.9725 ± 0.0018	0.9618 ± 0.0024	0.9671 ± 0.0017
LogSD	0.7807 ± 0.0049	1.0000 ± 0.0000	0.8769 ± 0.0035	0.8959 ± 0.0031	0.9986 ± 0.0004	0.9445 ± 0.0024
MIDLog	0.9684 ± 0.0018	0.9625 ± 0.0021	0.9654 ± 0.0016	0.9926 ± 0.0007	0.9881 ± 0.0010	0.9903 ± 0.0006
PulseLog	0.9846 ± 0.0013	0.9891 ± 0.0010	0.9868 ± 0.0009	0.9908 ± 0.0008	0.9917 ± 0.0007	0.9912 ± 0.0006

across training budgets suggests that PulseLog remains effective as the amount of labeled training data varies.

On HDFS, as shown in Table 18 and Table 19, across 5K–20K, HDFS F1 improves for PulseLog from 0.9868 to 0.9951, with low reported dispersion (SD 0.0003–0.0009). The best performance is not uniform across metrics: PLELog achieves the highest precision at 5K–15K, while LogDLR leads at 20K. LogSD reports the highest recall across all four budgets, highlighting that the F1 ranking reflects the balance between precision and recall rather than any single metric. The monotonic

Table 19: HDFS results for 15K and 20K training size

	15K			20K		
Method	P	R	F1	P	R	F1
DeepLog	0.9515 ± 0.0024	0.9348 ± 0.0029	0.9431 ± 0.0022	0.9685 ± 0.0018	0.9526 ± 0.0024	0.9605 ± 0.0019
GAT	0.9816 ± 0.0013	0.9847 ± 0.0012	0.9831 ± 0.0011	0.9885 ± 0.0010	0.9902 ± 0.0008	0.9893 ± 0.0009
KNN	0.6402 ± 0.0074	0.0822 ± 0.0058	0.1457 ± 0.0064	0.8736 ± 0.0045	0.4579 ± 0.0067	0.6010 ± 0.0052
LogAnomaly	0.9815 ± 0.0012	0.9726 ± 0.0018	0.9770 ± 0.0013	0.9912 ± 0.0008	0.9825 ± 0.0014	0.9868 ± 0.0009
LogDLR	0.9946 ± 0.0006	0.9885 ± 0.0009	0.9915 ± 0.0006	0.9964 ± 0.0004	0.9905 ± 0.0007	0.9934 ± 0.0004
LogFormer	0.9691 ± 0.0019	0.9467 ± 0.0026	0.9578 ± 0.0020	0.9875 ± 0.0010	0.9774 ± 0.0015	0.9824 ± 0.0011
Lograph	0.9934 ± 0.0006	0.9891 ± 0.0009	0.9912 ± 0.0006	0.9957 ± 0.0004	0.9918 ± 0.0007	0.9937 ± 0.0004
NeuralLog	0.8648 ± 0.0034	0.9745 ± 0.0016	0.9165 ± 0.0026	0.9125 ± 0.0028	0.9718 ± 0.0017	0.9412 ± 0.0023
PCA	0.4368 ± 0.0061	0.9756 ± 0.0014	0.6033 ± 0.0043	0.5005 ± 0.0057	0.9695 ± 0.0016	0.6641 ± 0.0040
PLELog	0.9952 ± 0.0004	0.9904 ± 0.0008	0.9928 ± 0.0004	0.9961 ± 0.0003	0.9927 ± 0.0006	0.9944 ± 0.0003
RobustLog	0.9862 ± 0.0010	0.9715 ± 0.0018	0.9788 ± 0.0011	0.9915 ± 0.0007	0.9736 ± 0.0015	0.9825 ± 0.0009
LogSD	0.9524 ± 0.0020	0.9975 ± 0.0005	0.9744 ± 0.0015	0.9830 ± 0.0012	0.9968 ± 0.0005	0.9898 ± 0.0008
MIDLog	0.9948 ± 0.0005	0.9912 ± 0.0008	0.9930 ± 0.0005	0.9958 ± 0.0004	0.9935 ± 0.0006	0.9946 ± 0.0004
PulseLog	0.9936 ± 0.0005	0.9932 ± 0.0006	0.9934 ± 0.0004	0.9954 ± 0.0004	0.9948 ± 0.0005	0.9951 ± 0.0003

Table 20: Hadoop results for 5K and 10K training size

	5K			10K		
Method	P	R	F1	P	R	F1
DeepLog	0.6889 ± 0.0054	0.6889 ± 0.0051	0.6889 ± 0.0048	0.9800 ± 0.0015	0.9455 ± 0.0028	0.9624 ± 0.0018
GAT	0.9712 ± 0.0018	0.9473 ± 0.0024	0.9591 ± 0.0015	0.9865 ± 0.0012	0.9885 ± 0.0010	0.9875 ± 0.0009
KNN	0.5188 ± 0.0072	0.6479 ± 0.0061	0.5762 ± 0.0064	0.5446 ± 0.0068	0.6579 ± 0.0059	0.5959 ± 0.0061
LogAnomaly	0.9765 ± 0.0015	0.9514 ± 0.0022	0.9638 ± 0.0013	0.7860 ± 0.0043	0.8243 ± 0.0038	0.8047 ± 0.0035
LogDLR	0.9878 ± 0.0010	0.9525 ± 0.0018	0.9698 ± 0.0011	0.9968 ± 0.0005	0.9865 ± 0.0010	0.9916 ± 0.0006
LogFormer	0.8893 ± 0.0034	0.8603 ± 0.0037	0.8743 ± 0.0030	0.9237 ± 0.0028	0.8905 ± 0.0033	0.9068 ± 0.0027
Lograph	0.9643 ± 0.0020	0.9496 ± 0.0023	0.9569 ± 0.0017	0.9689 ± 0.0018	0.9484 ± 0.0024	0.9586 ± 0.0016
NeuralLog	0.7624 ± 0.0049	0.7099 ± 0.0052	0.7348 ± 0.0044	0.8047 ± 0.0042	0.7608 ± 0.0047	0.7816 ± 0.0040
PCA	0.6238 ± 0.0062	0.5754 ± 0.0068	0.5982 ± 0.0059	0.6081 ± 0.0065	0.5638 ± 0.0071	0.5847 ± 0.0062
PLELog	0.9854 ± 0.0012	0.9586 ± 0.0018	0.9718 ± 0.0010	0.9972 ± 0.0004	0.9875 ± 0.0009	0.9923 ± 0.0005
RobustLog	0.9175 ± 0.0032	0.8535 ± 0.0038	0.8844 ± 0.0029	0.9749 ± 0.0019	0.9115 ± 0.0029	0.9422 ± 0.0022
LogSD	0.9847 ± 0.0013	0.9558 ± 0.0019	0.9700 ± 0.0011	0.9990 ± 0.0002	0.9842 ± 0.0010	0.9915 ± 0.0005
MIDLog	0.7384 ± 0.0051	0.8237 ± 0.0042	0.7787 ± 0.0045	0.9432 ± 0.0027	0.9448 ± 0.0025	0.9440 ± 0.0021
PulseLog	0.9865 ± 0.0011	0.9685 ± 0.0014	0.9774 ± 0.0009	0.9992 ± 0.0002	0.9879 ± 0.0008	0.9935 ± 0.0004

Table 21: Hadoop results for 15K and 20K training size

	15K			20K		
Method	P	R	F1	P	R	F1
DeepLog	0.9957 ± 0.0006	0.9561 ± 0.0022	0.9755 ± 0.0013	0.9970 ± 0.0004	0.9777 ± 0.0017	0.9873 ± 0.0009
GAT	0.9901 ± 0.0008	0.9865 ± 0.0011	0.9883 ± 0.0007	0.9964 ± 0.0004	0.9889 ± 0.0008	0.9926 ± 0.0005
KNN	0.5830 ± 0.0063	0.5208 ± 0.0071	0.5501 ± 0.0066	0.6163 ± 0.0058	0.4470 ± 0.0074	0.5182 ± 0.0067
LogAnomaly	0.9968 ± 0.0005	0.9865 ± 0.0010	0.9916 ± 0.0006	0.9976 ± 0.0003	0.9889 ± 0.0008	0.9932 ± 0.0004
LogDLR	0.9985 ± 0.0003	0.9865 ± 0.0009	0.9925 ± 0.0005	0.9994 ± 0.0002	0.9877 ± 0.0008	0.9935 ± 0.0004
LogFormer	0.9508 ± 0.0021	0.9187 ± 0.0026	0.9345 ± 0.0020	0.9718 ± 0.0016	0.9520 ± 0.0022	0.9617 ± 0.0017
Lograph	0.9970 ± 0.0004	0.9836 ± 0.0012	0.9903 ± 0.0006	0.9965 ± 0.0004	0.9865 ± 0.0010	0.9915 ± 0.0006
NeuralLog	0.8715 ± 0.0034	0.8220 ± 0.0039	0.8463 ± 0.0033	0.9895 ± 0.0010	0.9723 ± 0.0015	0.9808 ± 0.0011
PCA	0.6317 ± 0.0058	0.5940 ± 0.0064	0.6125 ± 0.0056	0.6195 ± 0.0061	0.5849 ± 0.0067	0.6018 ± 0.0058
PLELog	0.9982 ± 0.0003	0.9878 ± 0.0008	0.9930 ± 0.0004	0.9992 ± 0.0002	0.9889 ± 0.0007	0.9940 ± 0.0003
RobustLog	0.9905 ± 0.0010	0.9192 ± 0.0026	0.9535 ± 0.0016	0.9975 ± 0.0004	0.9174 ± 0.0028	0.9558 ± 0.0017
LogSD	0.9988 ± 0.0003	0.9865 ± 0.0009	0.9926 ± 0.0005	0.9988 ± 0.0002	0.9871 ± 0.0009	0.9929 ± 0.0004
MIDLog	0.9766 ± 0.0015	0.9952 ± 0.0006	0.9858 ± 0.0009	0.9813 ± 0.0014	0.9982 ± 0.0004	0.9897 ± 0.0007
PulseLog	0.9991 ± 0.0002	0.9908 ± 0.0007	0.9949 ± 0.0003	0.9987 ± 0.0003	0.9968 ± 0.0005	0.9977 ± 0.0003

improvement in F1 with increasing training data also indicates that PulseLog can effectively exploit additional labeled examples without sacrificing stability.

On Hadoop, as shown in Table 20 and Table 21, PulseLog achieves the highest F1 at all training sizes, with SDs no larger than 0.0009. The best precision and recall vary across budgets: LogDLR achieves the highest precision at 5K and 20K, GAT at 10K recall, and MIDLog at 15K and 20K recall. This pattern suggests that PulseLog's F1 advantage arises from a balanced precision–recall trade-off rather than consistent dominance on individual metrics. The low variation across training

Table 22: Thunderbird results for 5K and 10K training size

	5K			10K		
Method	P	R	F1	P	R	F1
DeepLog	0.8777 ± 0.0046	0.8060 ± 0.0053	0.8403 ± 0.0042	0.9929 ± 0.0008	0.9800 ± 0.0014	0.9864 ± 0.0007
GAT	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
KNN	0.4587 ± 0.0071	0.8389 ± 0.0049	0.5931 ± 0.0062	0.4733 ± 0.0068	0.7190 ± 0.0057	0.5708 ± 0.0060
LogAnomaly	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
LogDLR	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
LogFormer	0.8426 ± 0.0041	0.8847 ± 0.0038	0.8631 ± 0.0034	0.9018 ± 0.0031	0.9385 ± 0.0026	0.9198 ± 0.0027
Lograph	0.9277 ± 0.0032	0.8907 ± 0.0037	0.9088 ± 0.0029	0.9971 ± 0.0004	0.9959 ± 0.0005	0.9965 ± 0.0003
NeuralLog	0.8165 ± 0.0047	0.8612 ± 0.0042	0.8383 ± 0.0039	0.8789 ± 0.0036	0.9224 ± 0.0031	0.8991 ± 0.0032
PCA	0.6117 ± 0.0061	0.5810 ± 0.0067	0.5959 ± 0.0058	0.6198 ± 0.0059	0.5885 ± 0.0063	0.6037 ± 0.0055
PLELog	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
RobustLog	0.9177 ± 0.0052	0.8207 ± 0.0050	0.8665 ± 0.0043	0.9890 ± 0.0010	0.8954 ± 0.0033	0.9398 ± 0.0022
LogSD	1.0000 ± 0.0000	0.9830 ± 0.0012	0.9914 ± 0.0007	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
MIDLog	0.8448 ± 0.0045	0.8100 ± 0.0051	0.8271 ± 0.0040	0.9988 ± 0.0003	0.9971 ± 0.0005	0.9979 ± 0.0003
PulseLog	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000

Table 23: Thunderbird results for 15K and 20K training size

	15K			20K		
Method	P	R	F1	P	R	F1
DeepLog	0.9935 ± 0.0006	0.9918 ± 0.0007	0.9926 ± 0.0005	0.9982 ± 0.0003	0.9988 ± 0.0002	0.9985 ± 0.0002
GAT	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
KNN	0.5126 ± 0.0063	0.6226 ± 0.0059	0.5623 ± 0.0058	0.5322 ± 0.0061	0.5532 ± 0.0064	0.5425 ± 0.0057
LogAnomaly	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
LogDLR	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
LogFormer	0.9345 ± 0.0024	0.9618 ± 0.0019	0.9479 ± 0.0021	0.9568 ± 0.0018	0.9786 ± 0.0012	0.9676 ± 0.0015
Lograph	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
NeuralLog	0.9126 ± 0.0029	0.9475 ± 0.0025	0.9297 ± 0.0026	0.9415 ± 0.0023	0.9678 ± 0.0018	0.9545 ± 0.0020
PCA	0.6042 ± 0.0062	0.5719 ± 0.0068	0.5874 ± 0.0059	0.6261 ± 0.0057	0.5960 ± 0.0064	0.6106 ± 0.0056
PLELog	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	0.9988 ± 0.0003	1.0000 ± 0.0000	0.9994 ± 0.0002
RobustLog	0.9835 ± 0.0014	0.9112 ± 0.0028	0.9460 ± 0.0020	0.9793 ± 0.0016	0.9195 ± 0.0025	0.9485 ± 0.0018
LogSD	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	0.9994 ± 0.0002	0.9997 ± 0.0001
MIDLog	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
PulseLog	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000

budgets further indicates that the observed advantage is not driven by a single favorable training regime.

On Thunderbird, as shown in Table 22 and Table 23, several methods achieve an F1 of 1.0000 at 5K, with the number of saturated methods increasing as the training size grows. PulseLog maintains perfect F1 across all training budgets, indicating that the benchmark is effectively saturated even under limited training data. Accordingly, Thunderbird provides limited discrimination among methods at higher training budgets, making the smaller-budget results more informative for assessing differences in data efficiency.