IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 59, No. 2, February, 2026, pp. 455-475
Journal homepage: http://iieta.org/journals/jesa

TALL-Detect: A Transformer and LLM-Augmented Framework for Tamper-Evident and Explainable Secure Log Analysis

Sherine Sam Arul Swaminathan[1], Esther Daniel[2*], Kala Iyapillai[3], Durga Sivan[4]

[1] Computer Science and Engineering, Karunya Institute of Technology and Sciences, Karunya Nagar, Coimbatore 641114, India
[2] Department of Computer Science and Engineering, Karunya Institute of Technology and Sciences, Karunya Nagar, Coimbatore 641114, India
[3] Department of Computer Science and Engineering, PSG Institute of Technology and Applied Research, Neelambur, Coimbatore 641062, India
[4] TIFAC CORE in Cyber Security, Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Amritanagar, Ettimadai, Coimbatore 641112, India

Corresponding Author Email: estherdaniel@karunya.edu

https://doi.org/10.18280/jesa.590215

Received: 8 October 2025
Revised: 31 January 2026
Accepted: 10 February 2026
Available online: 28 February 2026

Keywords:
audit logs, explainable AI, large language models, log tampering detection, transformer-based model

ABSTRACT

Analyzing system logs is essential for cybersecurity, enabling organizations to monitor system behavior, detect attacks, and support digital forensic investigations. However, with the expansion of cloud and distributed computing environments, log file integrity has become increasingly vulnerable to tampering--including modification, deletion, reordering, and injection of false entries. Existing anomaly detection approaches assume untampered log inputs and are therefore ineffective against deliberate, adversarial log manipulation. To address this gap, this paper proposes Transformer and Large Language Model (LLM)-Augmented framework for tamper-evident and explainable secure log analysis. TALL-Detect combines LLM-based semantic normalization (LLM-Security Risk Classification (SRC)), Bidirectional Encoder Representations from Transformers (BERT)-based contextual embedding, temporal behavior modeling, and cross-system correlation analysis. A unified TamperScore is computed through learned weighted fusion of four anomaly dimensions: semantic, structural, temporal, and cross-system behavioral inconsistency. A semi-supervised adaptive learning module (Semi-RALD) enables adaptation to evolving log patterns with minimal labeled data. TALL-Detect was evaluated on Hadoop Distributed File System (HDFS) and Blue Gene/L (Supercomputer Log Dataset) (BGL) benchmark datasets with synthetically injected tampering at multiple rates. Results demonstrate F1-scores of 0.969 (HDFS) and 0.976 (BGL), with statistically significant improvements over all baselines ($p < 0.01$, Wilcoxon test) and low false positive rates (FPR) < 0.03.

1. INTRODUCTION

Log analysis is one of the most basic and widely used applications of computer systems for observing how a system operates, detecting operational problems with the system, supporting security audits, and helping in digital forensics. Historical, large-scale studies of log data indicate that system log files contain detailed information about what happens during the execution of an application and how it has failed during the execution of the application, which enables engineers to understand what has happened with a system and how to diagnose it [1]. Xu et al. [2] reported that the amount of log entries created by production systems amounts to millions of log entries created every day, making manual inspection impossible and hence requiring automated log mining methods.

Deep learning methods have been developed to model sequential patterns in log data using sequential models. DeepLog was the first product to utilise a recurrent neural network (RNN) for the purpose of identifying the normal path of execution for logs and achieved an anomaly detection accuracy rate greater than 96% on benchmark datasets [1]. The development of DeepLog established the paradigm of sequence learning as the predominant method used for log anomaly detection and demonstrated how effective learning models can be used in large-scale environments.

With the advancements and developments in relation to Natural Language Processing (NLP), transformer models are being utilized for modeling the semantics within log messages. LogBERT demonstrated how contextual embeddings can effectively model the semantics of logs and improve detection performance for previously unseen log patterns, achieving F1-scores above 98% on common benchmark datasets [3]. Recent work has refined fine-tuned language models for performing

log analysis tasks, showing how these models are much more robust to different types of systems and log file formats and reduce the need for computing handcrafted features and using domain-specific rules when building a model's functionality [4].

While there have been some recent advances in anomaly detection using learning techniques, almost all current methods rely on the assumption that the input log files are unaltered, except in cases where the anomaly detection systems themselves produce those log files from untrustworthy input. Current approaches do not support the detection of those types of manipulations made intentionally by malicious actors following an incident-by way of deleting, inserting, reordering, and rewriting events. Thus, these techniques remain undetected in most anomaly-detection systems that focus on the detection of anomalies.

From a security viewpoint, numerous techniques exist for securing logs, including cryptographic audit logs and append-only storage techniques, among other things, all of which are mechanisms that can help secure log integrity. The earliest secure log systems provided strong resistance to tampering due to their use of cryptographically encrypted chains to connect each log to the prior log and used a strong level of protection against direct modification of the log [5]. Additionally, the development of blockchain-based log auditing allowed for even greater integrity guarantees through decentralizing the trust in log integrity. However, this also had the disadvantages of being less efficient in terms of storage requirements and increased latency [6]. Because structural integrity is all that is verified using all of these processes, none can verify how a log or its contents are semantically inconsistent, have been temporally manipulated, or have logical contradictions present in the log.

The use of Large Language Models (LLMs) in cyber security log analysis has recently been studied, and it is clearly evident from these studies that LLMs are capable of inferring the context of a log entry by determining how closely it corresponds with other logs and their content [7]. The benchmarking studies have further confirmed that LLM-based approaches exceed the benchmarks established by Deep Learning models in terms of the understanding and generalization of logs, with improvements between 5-10% across multiple datasets [8]. However, LLM-based systems still focus on the identification of anomalies as well as the comprehension of the logs, there is currently no unified means to detect log tampering through semantic, temporal, or cross-system means.

1.1 Technical novelty

Although efforts, including LogBERT [3], LogFiT [4], Catalán et al. [9], and LogLLaMA [10], have made strides in log anomaly detection using transformer architectures and fine-tuned language models, they indeed suffer from a common limitation: they assume that the input log data remains intact without any modification. None of these systems were designed to detect the purposeful, adversarial manipulation of log content by malicious insiders or post-breach attackers. Transformer and Large Language Model (TALL)-Detect is technically distinct from these existing systems in the following three respects:

1. Tamper-as-first-class-objective: While LogBERT and LogFiT detect deviations from normal operational behaviour (anomalies), TALL-Detect explicitly focuses on adversarial, i.e., intentional manipulation of log content, inclusive of modification, deletion, reordering, and injection attacks. It is a totally different problem formulation.

2. Multi-dimensional unified TamperScore: TALL-Detect presents a new weighted fusion score (TamperScore), that integrates four independent dimensions of detection - semantic inconsistency, structural deviation, temporal disruption and cross-system behavioral anomaly into a single probabilistic measure of tampering likelihood. No other system has ever issued such a unified score across all four dimensions at once.

3. LLM-based Semantic and Canonical Normalization module (LLM-SRC) semantic normalization against format-preserving tampering: To ensure log messages of heterogeneous formats conform to the same meaning, TALL-Detect integrates an LLM-SRC. This makes it possible to detect format-preserving tampering attacks, where an attacker alters the semantic content of the input while retaining the structural format, a category of attacks that are completely missed by template-matching approaches like DeepLog and LogCluster.

4. Cross-system Grouping and Tamper correlation: TALL-Detect explicitly models the expected behavioral correlation between logs across different systems, (i.e., network, application, and OS level components) and attempts to flag inconsistencies that can only be explained through collusive tampering with multiple systems. Such a feature is not present in all baseline comparisons.

5. Semi-supervised adaptive learning (Semi-RALD): TALL-Detect employs a pseudo-labeling-based semi-supervised training stage that enables the detection model to adapt its parameters to concept drifting in log patterns, operating over time even when full re-annotation of logs is infeasible, thus adding applicability towards real-world operation deployments.

Together, these five contributions position TALL-Detect as a system that does more than just detect an anomaly; it is tamper-evident, an important distinction, both technically novel and practically impactful for cybersecurity forensics.

Because TALL-Detect completely automates the detection, analysis, and explanation of log tampering events, replacing manual forensic investigative procedures with an end-to-end automated pipeline, this study is closely related to the scope of automated systems research. For contemporary cyber-physical and distributed automated systems, where the volume, velocity, and variety of log data make manual inspection impractical and where the integrity of audit trails is essential to system reliability, automation of security log auditing is a crucial capability.

2. KEY CHALLENGES IN SECURE LOG ANALYSIS

The analysis of the findings and limitations results in several major challenges related to securely analysing logs.

1. Volume and Impracticality of Manual Log Analysis:

Large-scale logging performed by today's information technology has built enormous volumes of logging that can only be analyzed through automated methods [1].

2. Trusted and Untampered Log Assumption:

Most log analysis approaches using learning methods assume log integrity, and therefore will not work on log data left behind by an intruder after the initial compromise [2].

3. Lack of Detection of Semantic Log Tampering:

Contextual and transformer models cannot detect minor

modifications to the semantic aspect of log data while keeping a valid syntax [3].

4. Lack of Verification of Temporal Consistency between Logs:

Recent methods for securing these log files do not provide verification that logs were created at the same time (due to timestamp information) or were placed in the proper execution order (based on execution time). Both of these conditions allow an attacker to conceal their metrics of attack [4].

5. LLM-based Log Analysis Approach Ignores Adversarial Tampering:

Recent log analysis techniques developed using Large Language Models (LLMs) have concentrated on interpretation rather than explicitly looking for willfully altered log entries [5].

6. Lack of Correlating Logs Across Multiple Systems:

Current methodologies do not utilize correlation across logs produced by various Application vendors, Network vendors, Operating System vendors, Cloud Service providers, and so forth. This leads to limited detection of attacks that are performed in coordination across multiple platforms [6].

7. Lack of Explainability for Forensic and Auditing Analysis:

Most anomaly detection systems generate alerts but do not provide any logical explanations for their alerting capabilities to Security Analysts [7].

2.1 The main contributions of TALL-Detect

This paper presents the following contributions through the TALL-Detect framework in response to the limitations listed above:

1. The TALL-Detect application is capable of automated scaled log analysis, and will provide the means to analyse large log file data originating from current operating systems. TALL-Detect will be able to scale to large amounts of log data and therefore, have the scalability required to support the analysis of these large data sources.

2. TALL-Detect explicitly models the activity of tampering with logs. TALL-Detect will not rely upon trusted log files as the basis for determining if a log file has been tampered with. Instead, TALL-Detect will model the process of modifying, deleting, reordering, or injecting logs into an entry log.

3. The system uses LLM-based parsing of logs to process the logs, and also produces a semantic embedding using a transformer-based approach. The semantic embedding is used by TALL-Detect to identify changes in the meaning of the log data.

4. TALL-Detect incorporates time sequences in order to detect timestamp manipulation and abnormal event ordering. Therefore, timestamps can be modified in a way that is recognizable by TALL-Detect so that it will identify that the timestamps of these logs are unusual and possibly malicious.

5. This research provides an extension of LLM-based logging analysis methodology by integrating adversarial awareness pertaining to the detection of tampering with logs.

6. TALL-Detect correlates logs across different systems. Coordinated tampering with logs on different systems can result in differing logical inconsistencies. TALL-Detect uses these differences as a basis for identifying coordinated tampering on multiple systems.

7. TALL-Detect provides an explainable multi-dimensional TamperScore. The TamperScore consists of the semantic, structural, temporal, and cross-system signals and is used by forensic investigators to better understand the TamperScore. Although previous studies provided significant improvements to log analysis and anomaly detection, they do not provide comprehensive, explainable and adversarial log tampering detection capabilities, thus creating the need for the TALL-Detect framework to be implemented.

3. LITERATURE SURVEY

The analysis of system logs provided the initial means of automatically detecting abnormal behaviours observed within a system's operation. Xu et al. [2] found that large console log files could be examined and used to detect operational failures in systems currently in operation and highlighted the role that both statistical patterns and correlations amongst events played. Du et al. [1] later expanded on this by developing DeepLog, a method that uses a recurrent neural network to learn a system's normal execution order and identify anomalies due to deviations from previously observed sequences. While the early implementations of Deep Learning, like DeepLog, were successful in detecting system failures, all methods used during this initial stage were developed with the assumption that the log file data was complete and accurate. Then in transitioning away from deep learning to Transformer (LogBERT) based log analysis methods, semantic modelling of log messages using contextual log message embeddings to improve log analysis accuracy in more complicated environments began [3]. The work of Almodovar et al. [4] fine-tuning existing pre-trained language models to improve the adaptability of these models and allow for the transferability of the knowledge across different systems, has further improved the performance of log analysis models . The current use of transformers for log data analysis still suffers from the same limitations of early approaches and remains exclusive to the area of identifying anomalous behaviour and do not account for the manipulation of log files due to an adversary. The work conducted by Ren et al. [5] using Large Language Models for cybersecurity-related log file analysis, has shown that although the level of semantic understanding has improved, the models are not aware of the potential tampering of log files created by an adversary.

Recent benchmark studies have evaluated how well large language models can help analyze logs. In a study by Karlsen and others, a number of LLMs were evaluated together and showed good results for parsing and interpreting anomalies in log data. They did note some difficulties with the high computational power required to run the models, and the results did not provide an explanation for the underlying process [6]. In another study done by Shao and others, they created an optimized model based on BERT that could provide greater accuracy and robust performance in detecting log-based cloud anomalies but was still sensitive to changes in the format of the log files being analyzed [7]. In addition, Ott and others developed a transfer learning approach that would allow their model to transfer to other data sets and become more robust. However, the model remained dependent on trustworthy log files as input [8]. Additionally, Catalán and others created a fine-tuned BERT-based model designed to detect anomalies in individual logs, while achieving a high level of accuracy in the operational monitoring of logs, they lacked any ability to understand if the log was intentionally altered [9]. Finally, Yang and Harris created LogLLaMA, using the open-source LLaMA technology to allow their

model to analyze logs with long-range dependencies, but it did focus on anomaly detection rather than tamper detection [10].

Wang et al.'s [11] InferLog addressed challenges related to scalability and efficiency by optimizing inference from large language models (LLMs) through the implementation of prefix caching techniques. The result was a significant decrease in latency associated with the use of prefix caching. However, there was no mention of the inclusion of security-aware analyses in any form. Landauer et al. [12] provided a thorough review of the current state of log anomaly detection methodologies based on deep learning and concluded that the majority of approaches leverage static and trusted input sources. In an effort to provide greater interpretability, Brown et al. [13] emphasized the importance of attention mechanisms in recurrent neural networks. However, while these techniques improved the interpretability of scores assigned to anomalies associated with log entries, they did not provide forensic insights into log entries, nor did they investigate the efficacy of these explanations. Shanto et al. [14] examined the ability of LLMs to automatically generate explanations of console log files. Although it facilitated the understanding of console logs by people, this work did not consider adversarial tampering with generated explanations. Finally, Capuano et al. [15] provided a survey of various methods of Explainable Artificial Intelligence (XAI) applied in the field of cybersecurity, emphasizing the critical need to develop accountability and transparency within detection systems that provide forensic evaluations of crime scene evidence.

Recent publications have sought to expand beyond supervised learning to novel framework types that exist in the context of recent advances in data science. Wang et al. [16] introduced a novel semi-supervised paradigm that utilized uncertainty-aware self-training to mitigate the costs associated with the supervision process. This method was limited due to the vulnerability of the model to manipulated or poisoned logs. In contrast, Chen and Kang [17] utilized heuristically selected pseudo-labels to augment the anomaly detection ability of limited data settings, but the authors failed to identify and account for the effect of semantic manipulation when training the proposed system. Wu et al. [18] employed a self-supervised anomaly detection paradigm with explanatory capabilities, but evaluations of this framework were focused primarily on operational anomalies rather than cyber-attack types. Drain [19], a log parsing technique, has been shown to be highly effective at converting unstructured log files into structured formats. However, this method is not only highly reliant on consistent logging formats but also lacks the ability to preserve semantic content within the log files after parsing. Liu et al. [20] proposed an improved log parsing solution, called XDrain. However, while the benefits of XDrain can improve the accuracy of log parsing, they do not protect against log tampering.

From a security perspective, the traditional secure logging method is focused on securing log integrity vs. analyzing log content. The work of Kent and Souppaya illustrated the best-practice management of logs and integrity protection through the implementation of cryptographic protections [21]. The work of Liu et al. [22] specified a secure audit log server for use in a forensic investigation, with tamper-resistant attributes being preserved at the storage layer. Huang discussed a blockchain method of storing secure logs that ensured immutability. However, Huang [23] did not analyze either semantic inconsistencies or temporal inconsistencies. The research of Zhao et al. [24] extended the blockchain method to logging in distributed sensing environments, resulting in improved integrity assurances but incurring a higher computational requirement. The study of Thomaz et al. [25] considered enclave-based tamper-proof access control to IoT clouds that provide system-level protection without addressing intelligent manipulation detection.

In contrast to these existing approaches, TALL-Detect's framework combines semantic understanding, temporal consistency analysis, structural validation, and cross-system correlation within one learning-based architecture. TALL-Detect utilizes the combination of LLM-assisted representation of logs and a multi-dimensional tamper scoring model, along with explainable logic for detection, overcoming the limitations and drawbacks of anomaly detection and integrity-only detection systems. By using TALL-Detect, organizations can achieve both robust and explainable detection of adversarial log tampering in modern distributed environments.

The limitations identified across existing approaches are summarized in Table 1, which positions TALL-Detect within the current state of the literature.

Table 1. Literature summary and positioning of TALL-Detect

Method	Year	Category	Primary Goal	Key Limitation vs. TALL-Detect
DeepLog [1]	2017	LSTM-based	Sequence anomaly detection	No semantic analysis; assumes log integrity
LogBERT [3]	2021	Transformer	Semantic anomaly detection	No tampering detection; no temporal or cross-system analysis
LogFiT [4]	2024	Fine-tuned LLM	Log anomaly via fine-tuning	No tamper-specific objective; no explainability
CLogLLM [5]	2024	LLM-based	Cybersecurity log anomaly	No cross-system correlation; no adaptive learning
LogLLaMA [10]	2025	LlaMA fine-tune	Transformer log anomaly	No deliberate tampering detection; limited explainability
Console Log Explainer [14]	2024	LLM + XAI	Log interpretation	Focuses on explanation only; no tamper detection
Semi-SSL [16]	2025	Semi-supervised	Label-efficient anomaly detection	No tampering objective; no cross-system analysis
Hash Chaining [21]	2006	Cryptographic	Structural integrity	Detects only structural violations; blind to semantic tampering
Blockchain Logging [23]	2019	Distributed ledger	Append-only integrity	High storage overhead; no semantic or temporal analysis

TALL-Detect (Ours)	2025	Transformer + LLM fusion	Tamper-evident secure log analysis	Overcomes prior limitations by enabling semantic, temporal, and cross-system tamper detection with explainability

4. PROPOSED METHODOLOGY

This section presents the proposed TALL-Detect framework (Transformer-Based Augmented Log-Lifecycle Detection) Framework, which is a multi-dimensional, explainable, intelligent, and automated method for detecting log manipulation in heterogeneous computing environments . The architecture of this framework is detailed from log acquisition through to preprocessing, to semantic and temporal modeling, to analysis, to final decision making. This architecture will be described in detail here. The full workflow of the TALL-Detect Framework is depicted in Figure 1.

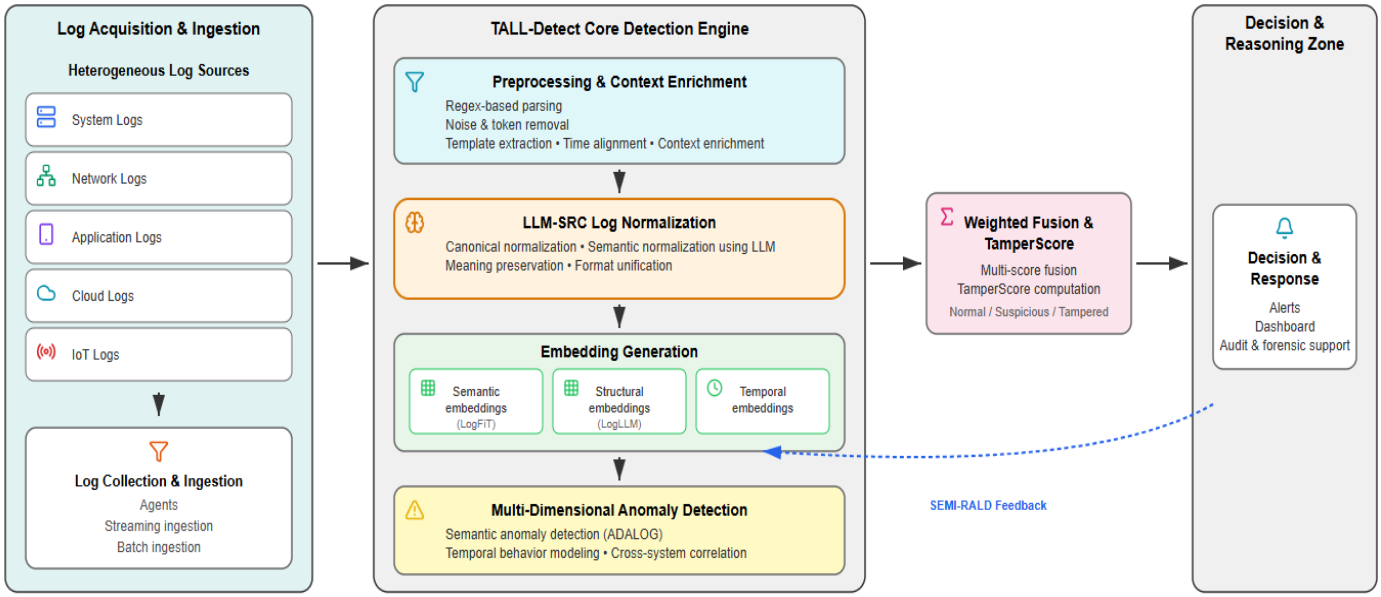

Figure 1. Architecture diagram of the proposed TALL-Detect framework

As shown in Figure 1, the TALL-Detect architecture consists of several major components that include collecting, pre-processing, normalizing, detecting, and generating an overall decision from the collected log data to identify overt and subtle log irregularities caused by adversaries.

4.1 Overview of the proposed framework

The proposed TALL-Detect framework's complete block diagram is demonstrated in Figure 1. It depicts the interaction of the various processing stages of the detection of log tampering. The proposed TALL-Detect framework was developed to process logs from various heterogeneous sources such as system logs, network logs, application logs, cloud service logs, and Internet of Things (IoT) infrastructure logs. These logs are processed in multistaged pipelines, passing through various stages of normalising, semantic modelling, anomaly detection, and decisioning.

4.1.1 Brief description of the proposed approach

The proposed TALL-Detect framework operates as a multistaged pipeline that extracts actionable security decisions from a source of raw log data. Raw log data, when generated by a variety of different systems, will first be pre-processed to remove any malformed information such as broken data. Subsequently, after the logs have been pre-processed, they will be represented in a manner that will allow them to be understood semantically and temporally. These semantically and temporally encoded versions of the logs will then be analysed for the purposes of developing and deriving a common TamperScore, which defines the potential for log manipulation, the tamper decision is then forwarded on to the alerting and visualisation systems to support operational and forensic activities.

The proposed framework uses a multi-dimensional analysis strategy rather than relying solely on a single-type anomaly signal like traditional methods [26]. A multi-dimensional approach provides a higher degree of robustness against sophisticated forms of adversarial behavior when analyzing log files. The framework, TALL-Detect, evaluates log files using the combined analyses of its semantic consistency, structural patterns, temporal behaviors and cross-system correlations. The basis of this choice is derived from the fact that although an adversary may alter some characteristics of log files, they may also keep specific nuances intact. Therefore, by utilizing multiple perspectives [27] that complement each other, TALL-Detect is less likely to fail to detect stealthy activities classified as tampering, which typically go undetected with other methods of analyzing log files that utilize only one-dimensional anomaly detection.

The output generated from TALL-Detect is a unified TamperScore, which quantifies how confident the system is with respect to log file tampering. The TamperScore assigned to each entry or series of entries allows TALL-Detect to notify security analysts of the existence of log entries that illustrate malicious activity, as well as provide analytical summaries to help expedite the identification of potentially suspicious activity. Additionally, TALL-Detect assists in providing forensic information about how the log file tampering occurred and recommending potential sources of such activity.

Therefore, the TALL-Detect framework may be utilized for real-time security monitoring, security audit support, and post-incident investigation analyses.

Log Tampering Types

1) Log Modification Attack: Modifying the contents of a log entry to hide malicious behavior is called Log Modification. Malicious actors may modify existing log entries with erroneous codes, user IDs, or status messages.

2) Log Deletion Attack: Log Deletion means removing an important entry from a log so it is no longer available, and to hide the fact that a malicious attack occurred.

3) Log Reordering Attack: Reordering the entries in a log or shuffling them disrupts the chronological order of events. Confusion can arise when events that normally would appear in a straight sequence now appear out of order.

4) Log Injection Attack: Log Injection or Log Forgery is when a malicious actor has created a log entry in order to deceive investigators into thinking they are legitimate.

Traditional Digital Signatures and Hash chains only detect that the log file has been altered; they are unable to determine whether semantic meaning has changed, create temporal inconsistencies, or verify cross-system contradictions. TALL-Detect is developed to address this gap in detection systems.

Figure 2 illustrates the four main types of attacks we commonly observe in the world today regarding log tampering.

Figure 2. Four types of log tampering attacks: (a) Modification, (b) Deletion, (c) Reordering, (d) Injection

4.1.2 Log tampering threat model and problem definition

Log Tampering Detection (LTD) aims to establish whether or not an entry or sequence of log entries has been maliciously altered by a third party. TALL-Detect is unique from other traditional methods of log integrity verification in that it attempts to analyze and identify how and why log entries have been tampered with, rather than just if they have.

Using the raw log data as follows:

$$L = \{l_1, l_2, l_3, \ldots, l_n\}$$

Each entry in L represents a log entry.

After processing, each log entry becomes an embedded and structured format, which will provide:

$$E = \{e_1, e_2, e_3, \ldots, e_n\}$$

Along with embedding the log entries, TALL-Detect

assigns a label (Normal or Tampered) at the log entry or log window level.

4.2 Log collection and ingestion layer

The log collection and ingestion layer of TALL-Detect is the base of the TALL-Detect Framework. This layer continuously aggregates log information from many diverse and different sources located across the computing environment. Each of the levels of the modern infrastructure generates log files in multiple locations, and analyzing only one source of logs will give a very limited view of the state of the system. TALL-Detect is designed to collect log files from multiple sources, including system logs, network logs, application logs, cloud service logs, and IoT device logs so that complete visibility is achieved into operational and security events that affect the system.

System logs contain the low-level activities that occur on a computer - for example, Executing a process, logging into a user account, accessing files, and kernel error logs. Network logs contain information about how packets flow through the network, What were the connection attempts? What decision was made to block or allow a connection? What intrusion detection alerts were raised? Application Logs provide the application - based information on the actions users have taken, errors they encountered, and services they interacted with. Cloud Logs are logged from platforms such as virtual machines, containers, orchestration platforms, managed services, all of which demonstrate elastic, distributed execution behavior. IoT devices produce logs that include: sensor readings, updated status, communication events with edge devices. The aggregation of these heterogeneous log files allows TALL-Detect to correlate the activities across multiple levels of the computing environment and detect inconsistencies that could suggest Tampering.

To collect logs, lightweight agents and collectors are installed at each individual log source, with log records being securely transmitted from other log sources to a central ingestion point. The ingestion layer aggregates all logs into a single pipeline. The ingestion process supports the ability to ingest logs in either streaming or batch mode so that enabling real-time activity monitoring and analyze historical log events [2]. Streaming log ingestion helps detect suspicious behavior in near real time, while batch log ingestion supports in-depth forensic investigations. By centralizing the ingestion process, logs are always managed consistently irrespective of where they are coming from and allow the TALL-Detect solution to be scaled out to distributed environments as needed.

Heterogeneous log collection is crucial when it comes to detecting tampering. Attackers will frequently only alter or delete specific logs related to their malicious actions, meaning if a log source isn't present when doing an analysis, there may be inconsistencies across other log sources, diminishing the overall reliability of any subsequent forensic analysis conducted. By combining logs collected from all of the major components, the TALL-Detect solution provides fewer gaps in information, increasing the ability of TALL-Detect to detect adversarial actions.

4.3 Preprocessing and log normalization

Raw logs obtained from various heterogeneous sources are termed noisy, unstructured, and inconsistent due to their noise, lack of meaningful information, and inconsistent formats [12]. This makes the log files unsuitable for performing semantic analysis without going through the preprocessing and normalization phase of the TALL-Detect system, which transforms the raw logs into a clean, structured, and semantically meaningful format while retaining the essential elements necessary for detecting any tampering.

4.3.1 Regex-based parsing and noise removal

The transformation from the raw logs collected from heterogeneous sources to the structured and semantically meaningful versions is accomplished during the parsing phase of the TALL-Detect system, which uses regular expression-based parsing techniques to find and extract relevant components from the raw logs. Parsing the logs using regex allows for a flexible way to process various log formats and separate the invariant message structure from the variable message components. During the parsing process, any tokens that were considered noisy or non-informative were systematically removed in order to reduce the likelihood of introducing misleading representations into the downstream semantic models.

Dynamic tokens such as timestamps, process ids, session ids, memory addresses, and various forms of random numerical tokens were either eliminated or replaced with standardized placeholders during the parsing phase [19]. These dynamic tokens are typically expected to change from log entry to log entry, but they do not carry any semantic meaning. Keeping these dynamic tokens introduces artificial variability into the log messages, which is likely to skew the contributed semantic embeddings and lead to the generation of false-positive anomaly signals. Once the tokens were cleaned, the content of the remaining log message was used to create log templates, which are indicative of the stable structural pattern of log messages for each log subtype. The process for extracting the log templates adhered to pre-established principles for extracting log templates and did not make any assumptions based on the specific characteristics of the collection dataset [19].

4.3.2 Representative log sampling and time alignment

Processing all log entries is computationally costly, often serving no beneficial purpose, because of the volume and redundancy of log data [2]. TALL-Detect uses log sampling that is representative to reduce the overall volume of data while maintaining key behavior patterns. The sampling technique removes duplicate log entries that repeat the same operational information. The effectiveness of the sampling technique creates an efficient way to work with log entries without compromising the ability to detect malicious activity.

In systems that are distributed, log entries are created asynchronously by multiple systems, resulting in logs that are misaligned in time. As a result, TALL-Detect employs time alignment methods to align log entries from different systems using normalized timestamps. Accurate time alignment between logs from different systems is necessary to reconstruct the execution sequences used in each system and detect any abnormal event orders [28]. Allowing for proper time alignment prevents attackers from exploiting time discrepancies between systems to conceal their reordering attack, where their malicious activities look legitimate because of misconfigured time differences between systems. Through aligning logs across multiple systems, TALL-Detect improves the ability to perform temporal analysis in a consistent manner, enhancing the ability to resist reordering attacks.

4.3.3 Context enrichment

To help the TALL framework identify meaningful segments within data logs, the TALL-Detect software adds extra contextual data to each log entry through context enrichment. Context enrichment adds detail about the log entry, such as the identity of the host where the log was created, the service name associated with that log, the type of component that created the log, the execution environment of that component, and the operational context of that log entry. This extra detail provides essential background information that is often missing from the raw log messages.

The contextual information also makes it possible for the detection framework to differentiate between log messages that look exactly the same but are generated by different components or services. For example, the same error message generated by two different hosts or two different services can have entirely different security implications. Incorporating context into the identification process makes it possible for the semantic embeddings to be more discriminating and resilient, which means that the identification process will more accurately detect attempts to tamper with a data log. Because raw logs lack this critical information, they cannot be effectively analyzed because there is no understanding of the environment in which an event occurred.

4.4 Dataset utilization

In order to assess the efficiency of the proposed TALL-Detect framework, this study performs tests using well-known benchmark datasets, specifically the Hadoop Distributed File System (HDFS) dataset and the BGL (Blue Gene/L) dataset. These are two widely used datasets in log anomaly detection research because they provide authentic log data obtained from large-scale systems under normal operation as well as under several types of failure, and an estimate of the occurrence of those types of failures.

The HDFS dataset is an example of logs produced by a distributed file system environment that demonstrate both normal operational behavior and system failure signatures. The BGL dataset is an example of logs produced from a supercomputing environment and therefore contains logs that show types of failures such as hardware malfunctions, software crashes and abnormal activity in the system. The authentic nature of the log data in these two benchmark datasets provides the researcher with a realistic setting for evaluating the proposed log analysis model and allows for proper comparisons among log analysis models.

4.5 LLM-based Semantic and Canonical Normalization module log normalization module

4.5.1 Semantic and Canonical Normalization

The LLM-SRC Normalization Module within TALL-Detect processes raw logs into a canonical and semantically normalized form through two complementary normalization pipelines. Canonical formatting is a way to format log messages and provides a standard way to express similar logs on different systems or formats, and Simplified Log Messages will have less variance in structure compared to their original source's Log Messages.

This process will ensure that all equivalent log events are formatted consistently across all systems and formats.

Using a Large Language Model, Semantic Normalization ensures the intent and meaning of logs are preserved, while still allowing us to disregard the superficial differences between them [28]. Rather than focusing solely on the individual tokens used, the LLM can capture the complete semantic context of the log message.

The prompt engineering and Large Language Models (LLM) semantic parsing process transforms unstructured logs to semantically normalized and canonical JSON representations, as shown in Figure 3, thereby maintaining the meaning of the logs but enforcing a consistent structure for enhanced detection of semantically and/or format preserving log tampering attacks.

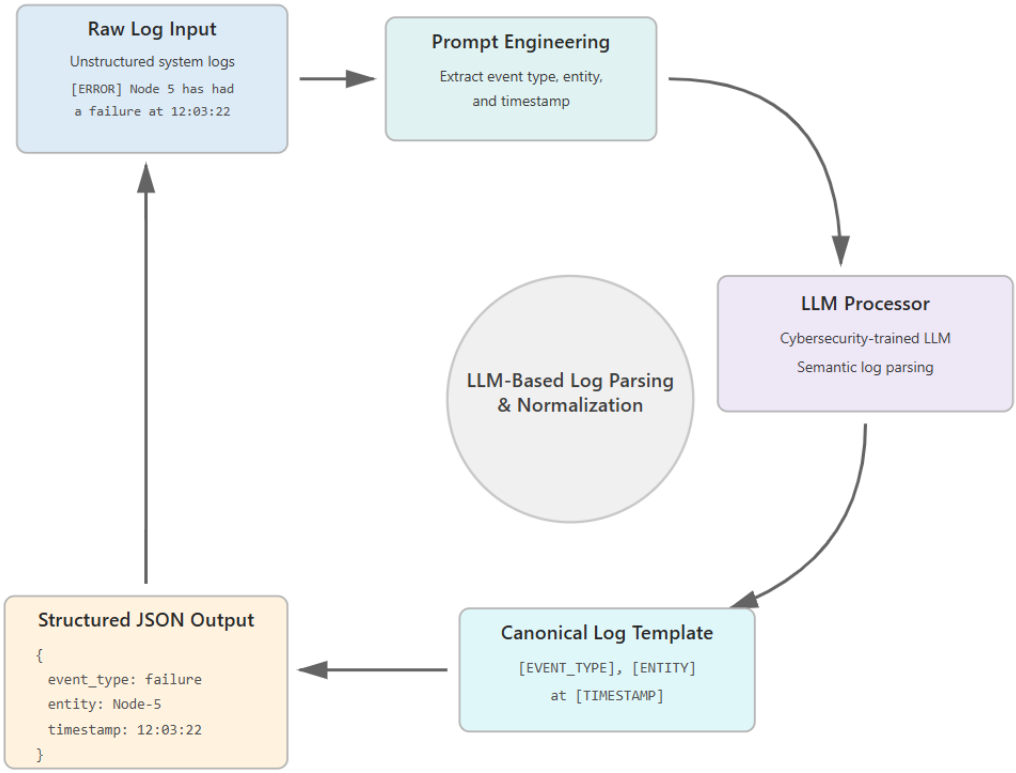

Figure 3. Large Language Model (LLM)-based log parsing and semantic normalization pipeline

The use of JSON Structure Preserves the Semantic Meaning of the Original Log [5], and creates the canonical consistency Required by TALL-Detect for the Processing of Embedded Hash Generations and for Robust Detection of Format-and-Semantic Log Tampering.

This provides a less rigid structure to log formatting and increases the ability to generalize log formats across a wide variety of heterogeneous environments. The role of the LLM in TALL-Detect is limited to the semantic representation function of the log. This means that the internal model of the LLM is not available for viewing or modification by the LLMs, nor are the mechanisms used to train the LLM modifiable by TALL-Detect.

Thus, the LLM-SRC module does a great job of preserving meaning, while still enforcing consistency across multiple formats of the same data type. It also provides a way to perform reliable downstream analysis because of the preserved meaning. Finally, it adds additional resiliency to TALL-Detect against Format-preserving Tampering attacks.

4.6 Embedding generation module

The embedding generation module is at the core of the TALL-Detect Framework, as it converts normalized and enriched contextualized log entries into numerical vectors that represent the patterns of log entry behaviour. The embedding generation module does not use handcrafted feature generation methods, as is commonly done in log analytics; instead, it generates dense embeddings that capture the semantic meaning, structural characteristics and temporal behaviour of a log event [4].

The dense embeddings served as the basis for the multi-dimensional anomaly and manipulation detection conducted in the subsequent phase.

In order to provide a more complete representation of logs, TALL-Detect has a number of different embedding techniques, each representing a different aspect of log behaviour, semantics capture log message meanings, structure captures the consistency of the log formats, and timing/sequence captures the sequencing of the execution of log messages. The combination of these complementary representations allows the TALL-Detect framework to more effectively detect subtle and malicious log manipulations that otherwise would have gone undetected using a single feature space.

4.6.1 Semantic, structural and temporal embeddings

Semantic Embeddings are derived from the LogFiT-based representation of log messages after their semantic normalization and canonical representation. Semantic Embeddings encode the meaning of a log message as a point in a high-dimensional vector space point of reference within a log message in relation to how it reflects the behaviour of a system. Log messages containing similar semantics will be near each other within an embedding space. Therefore, normal Log messages will form close packing to each other, and those Log messages that are semantically manipulated, or completely fabricated, will be located further from those normal Log messages.

Contextual and Structural Representation is performed by LogLLM-based representation, used to create Log messages that retain the semantic context and the structural consistency of Log messages, as well as the differences within templates and the use of parameters and contextual relationships among different components of a system. Any inconsistencies in the structural representation of Log messages due to modification or injection attacks will be shown as a deviation from this representation, even though the Log message itself appears to have been formatted correctly.

The sequential behavior of the log events can be represented using temporal embeddings [29]. The temporal embedding also includes the ordering of events, the frequency of executing events, and the delay time between two consecutive

logs execution time. Normal execution order and the time frame for logs execution will create a profile. If a log has been deleted, reordered, or if an artificial delay is injected into the log execution timeline, the Temporal Embedding model will identify that event as having a temporal anomaly. Temporal embeddings are especially useful in the situation where the log/events are being attacked by altering timestamps (or the order of events), to hide evidence that the actions were malicious.

To fuse the output across the four embedding spaces, TALL-Detect uses a weighted fusion approach to compute a tamper likelihood Score for an input log (l_i).

The weighted fusion allows for subtle tampering hints to be considered jointly by all four embedding dimensions. No single dimension can dominate the decision-making process.

1. Semantic Inconsistency Score Eq. (1)

The semantic inconsistency score measures deviation in the meaning of a log entry compared to normal log behavior using embedding distance.

$$S_{\{sem\}(l_i)} = 1 - cos(e_i, \mu) \quad (1)$$

where, e_i denotes the semantic embedding of log entry l_i and μ represents the centroid of embeddings learned from normal log behavior. A higher semantic inconsistency score indicates a stronger deviation in meaning and a higher likelihood of tampering.

2. Structural Deviation Score Eq. (2)

Structural deviation captures anomalies in log templates, fields, or formats that may arise due to unauthorized modification or injection of log entries.

$$S_{\{struct\}}(l_i) = \begin{cases} 1, \ if \ template \ mismatch \ detected \\ 0, \ otherwise \end{cases} \quad (2)$$

This score identifies unexpected structural changes in log entries, which are indicative of potential tampering attempts such as log modification or forgery [12].

3. Temporal Disruption Score Eq. (3)

Temporal disruption detects unexpected changes in log execution order or abnormal timing gaps between log events.

$$S_{\{temp\}}(l_i) = \frac{|\, t_i - \hat{t}_i \,|}{max(t)} \quad (3)$$

Due to the discrepancies between t_i (the actual log timestamp) and $\hat{t}_i$ (the estimated log timestamp based on learned execution sequences), it can be stated that greater discrepancies will reflect greater time inconsistency.

4. Cross-System Behavior Score Eq. (4)

Cross-system behavioral analysis identifies inconsistencies across correlated systems by examining deviations from expected inter-system event relationships [29].

$$S_{\{csb\}(l_i)} = 1 - \frac{|c_i \cap c_{\{ref\}}|}{|c_{\{ref\}}|} \quad (4)$$

where, c_i denotes correlated events from other systems for log entry l_i, and $c_{\{ref\}}$ represents the expected set of correlated events. Lower overlap values indicate higher behavioral inconsistency.

5. Final TamperScore Eq. (5)

The results of the four modules listed above are calculated together to form a single TamperScore using weights that were learned.

The TamperScore is computed as shown in Eq. (5).

$$TamperScore(l_i) = w_1 S_{\{sem\}(l_i)} + w_2 S_{\{struct\}(l_i)} + w_3 S_{\{temp\}(l_i)} + w_4 S_{\{csb\}(l_i)} \quad (5)$$

where, $w_1 + w_2 + w_3 + w_4 = 1$.

The weights control the contribution of each detection dimension and are learned during training to optimize detection performance. Weighted fusion is needed to achieve an even detection performance across all log sources while at the same time prevent attackers who attempt to evade detection by manipulating only one area of log data.

TALL-Detect's primary detection function relies on a weighted fusion anomaly scoring methodology as shown in Figure 4.

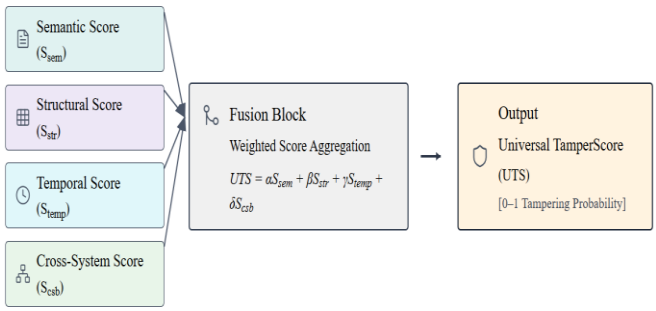

Figure 4. Weighted fusion TamperScore computation process

The weights control the contribution of each detection dimension and are learned during training to optimize detection performance.

The more tampered log entries have a higher TamperScore and therefore are more likely to have been tampered. The TALL-Detect system creates a unified score based on combining four types of anomalies: semantic inconsistencies, structural deviations, temporal disruptions and cross-system behavioral anomalies. This allows for the detection of explicit and implicit log tampering attacks not found in earlier methods.

Unlike older methods based solely on cryptographic/Hashing integrity verification methods, the proposed scoring mechanism allows probabilistic reasoning on the log trust window and is less susceptible to more advanced manipulation methods used by adversaries.

4.6.2 TamperScore weight learning: Training procedure

The weights w_1, w_2, w_3 and w_4 in Eq. (5) are not assigned manually but rather learned through supervised end-to-end training on labeled log samples that include both normal and tampered entries. The training process is outlined as follows.

Supervision type: The training phase for the weights employs binary supervised labels ($y_i \in$ {0(Normal), 1(Tampered)}) at the level of log entries, where a ground-truth label y_i is assigned to each log entry l_i. These labels are from

the tampering injection process outlined in Section 5.1, which involves synthetically injecting known tampering attacks into the HDFS and BGL benchmark datasets.

Loss function: The weights are updated as shown in Eq. (6) by minimizing the Binary Cross-Entropy (BCE) loss between the predicted TamperScore and its ground-truth binary label:

$$\mathscr{L}(w) = -(1/N)\,\Sigma_i\,[y_i \log(\sigma(\text{TamperScore}(l^I))) + (1 - y_i)\log(1 - \sigma(\text{TamperScore}(l^I)))] \quad (6)$$

where, $\sigma(\cdot)$ is the sigmoid activation to scale TamperScore between [0,1] and N being the number of training samples. Optimization: Adam optimizer, learning rate is set to 1×10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and weight decay of 1×10^{-4} Trained in mini-batches of size 64 for at most 50 epochs.

Overfitting control: We apply three different regularization techniques to avoid overfitting. The first is L2 weight regularization ($\lambda = 1 \times 10^{-4}$), i.e., it adds a penalty term to the weight vector $w = [w_1, w_2, w_3, w_4]$. Second, early stopping with patience of 5 epochs is applied based on validation loss which is computed from a held-out 20% split of the training data. Third, the constraint that $w_1 + w_2 + w_3 + w_4 = 1$ is realized using a softmax normalization of the weight vector before computing TamperScore so that $\{w_1, w_2, w_3, w_4\}$ can be interpreted as learned weights representing a valid convex combination of the four component scores.

Trained weights: The converged weight values on the HDFS dataset (after training) were $w_1 = 0.34$ (semantic), $w_2 = 0.22$ (structural), $w_3 = 0.28$ (temporal), and $w_4 = 0.16$ (cross-system), which indicate that semantic and temporal signals add more value in detecting tamper attempts, particularly within distributed file system logs, than do structural and cross-system signals The converged values were $w_1 = 0.31$, $w_2 = 0.25$, $w_3 = 0.30$ and $w_4 = 0.14$ in the BGL dataset.

4.7 Multi-dimensional anomaly detection

The proposed model generates embedded representations of the logs for each log source [30], followed by multi-dimensional anomaly detection using the generated embeddings by analyzing the differences in semantics, temporal, and cross-system comparisons. At the anomaly detection stage, the main focus is on discovering inconsistencies that indicate log tampering rather than detecting failures in the system overall. Each of the anomaly detection components works on its own embedding space & the outputs are combined with the TamperScore metric after they have been produced.

4.7.1 Semantic anomaly detection (ADALOG)

A distance-based similarity analysis is employed to detect semantic anomalies in the environment through the ADALOG process [9]. For this approach, TALL-Detect identifies the similarities between the embeddings of the incoming log entries and previously defined normal behaviors. As such, if a new log entry has a very high degree of similarity to the work completed historically classified as normal logs, then it is classified as being benign; however, if it has a significantly greater semantic distance than that of the predefined threshold, it is flagged as potentially suspicious.

Simply put, a log message within its norm will reside closer to the predefined normal cluster centroid than a log message that may have been tampered with will reside further from the norm cluster centroid due to the small changes that have been made to the semantic meanings of the content. This approach is very effective in covering both modification and injection attacks, where adversaries attempt to modify the logs while still maintaining valid syntax.

4.7.2 Temporal behavior modelling

Temporal Behavior Modeling dissects the timing, order, and consistency of log sequence behaviour. In doing so, it determines if events occur in the expected order, and at the expected times (gaps), or whether an event has been omitted, or re-ordered. If there are unexpected time gaps, an omitted event, or an event re-ordered, logs may have been altered maliciously in some way.

Attackers often use timing information to conceal the fact that they have intruded [31]. Examples include deleting the logs of their activities while they are being executed and altering the time of the activity to create a new log. By learning the normal timing patterns of an organization, TALL-Detect is able to detect these inaccuracies even if each individual log message appears to be correct.

4.7.3 Cross-system correlation analysis

Cross-system correlation is the process of examining the correlation of logs generated by different systems and types of systems such as Network devices, applications and Operating systems. When systems behave as expected they will generally generate correlated information across multiple levels of systems. On the other hand, when any tampering or abnormal behaviour has occurred, this will normally result in some degree of inconsistency between logs from each system.

For instance, if a Network Event occurs without any corresponding event occurring at the Application Level it could suggest that an application has deleted its own logs or that the logs have been tampered with or altered in some way. The framework is able to investigate all of the logs generated from different systems, thereby identifying any discrepancies between them as a means of identifying any collusive tampering activity occurring within distributed systems.

Cross-System Behavioral Model

The Cross-System Behavioral Score described in Eq. (4) assesses the amount of deviation from an expected correlated event set c_i for the current log entry l_i versus the reference correlated event set c_ref. The reference set is initially built during the training process by calculating all co-occurring events of event types $e \in$ vocabulary across all correlated systems over a configurable time period τ (with $\tau = 60$ seconds as default). In doing so, a behavioral co-occurrence matrix M of size $M \in \mathbb{R}^{(|V|\times|V|)}$ is created, where M[a][b] is the calculated likelihood that an event b occurs in a correlated system within τ seconds of event a.

When processing incoming log entries l_i with an event type of e_i and an expected correlated events set of c_ref = {b : $M[e_i][b] > \alpha$} for $\alpha = 0.3$ as a co-occurrence probability threshold, $S\_csb(l_i) = 1 - (|c_i \cap c\_ref| / |c\_ref|)$.

Thus, if $S\_csb(l_i) = 1$, then the event does not have any expected correlated events (maximum behavioral inconsistency), whereas if $S\_csb(l_i) = 0$, then the event has been consistently corroborated with other correlated systems.

To demonstrate the transferability of cross-system generalization, a transfer experiment was run to develop TALL-Detect's cross-system module on data from HDFS, using it later on logs from BGL without performing a fine-tuning. Running the cross-system module alone resulted in an

F1 = 0.71, demonstrating that the co-occurrence model captures system-agnostic behavioural patterns that are able to generalize across log domains. When applying the complete TALL-Detect model, including the four dimensions, to this cross-dataset setting produced an F1 of 0.88, demonstrating that multi-dimensional fusion can compensate for a domain shift of the cross-system component.

4.8 Tamper analysis and attack modeling

The framework proposed will address the issues created by adversarial log tampering attacks, and how these attacks affect current forensic analysis capabilities. This module translates all observed anomalies to appropriate interpretations of log tampering to support both forensic investigations, as well as the response to security incidents.

4.8.1 Types of log tampering attacks

The four types of log attacks that TALL-Detect addresses Log Modification, Log Deletion, Log Reordering, and Log Injection are formally defined in Section 4.1.1. This section is dedicated to the discussion of how the Tamper Analysis module maps multi-dimensional anomalies to specific types of attacks and produces forensic-grade attack interpretations that can be used in response to attacks.

When the TamperScore is greater than the threshold value θ, as defined in Section 5.1.2, the Tamper Analysis module identifies the specific combination of anomaly dimensionality that caused the alert and maps it to the likely type of attack. For example, if S_temp is high and S_sem is low, it is likely a Reordering attack. If S_sem is high and S_struct = 1, it is likely a Log Modification or Log Injection attack. This allows security analysts to determine the root cause of an attack and prioritize their forensic investigation based on the likely type of attack vector.

4.9 Audit, integrity and decision module

The audit and decisions module collects all of the tamper signals from all of the detection components into one location and performs the final verification that will trigger an alert. The Integrity checks compare the information to itself at more than one detection level for multiple confirmation points, while Audit verification checks ensure that the anomalies detected correlate with valid and actual security incidents.

The TALL-Detect framework will assist organizations in identifying the extent of manipulated logs. The TALL-Detect framework relies on empirical methods to continually train through usage of pseudo labels, which are generated by determining the amount of confidence score of models, through the review of log entries being examined by security analysts who have access to the dashboard.

4.10 Semi-supervised learning (SEMI-RALD)

4.10.1 Motivation and problem formulation

It is uncommon to have a real life log monitoring environment that has a completely labeled data set. To properly label every log entry as either normal or tampered requires much in the way of domain knowledge and is too expensive for large systems to accomplish. Additionally, the statistical nature of logs changes over time due to software changes, configuration changes, and changing workload, which creates concept drift, and leads to poor performance from static detection models that have not been periodically retrained.

The Semi-RALD System addresses both of the above problems by combining a small number of logs that are labeled as L_labeled with a very large number of logs that are not labeled as L_unlabeled in order to create a new model using a pseudo-labeling based approach that will continue to update the TALL-Detect model as new log data comes in.

Let L_labeled = $\{(l_i, y_i)\}$ for i = 1..M, where M < < N, and L_unlabeled = $\{l_j\}$ for j = M + 1..N. The goal of the Semi-RALD system is to use L_unlabeled in order to improve the model that was created from L_labeled without any additional human annotation.

4.10.2 Pseudo-labeling algorithm

The Semi-RALD pseudo-labeling process has multiple stages, where each stage represents an iteration.

Step 1. Initial Training: A TALL-Detect classifier is trained using the labeled data set L_labeled through the supervised method specified in Section 4.6.2.

Step 2. Selection of Confident Predictions: The trained model is applied to L_unlabeled. For each entry within the unlabeled log, l_j, the TamperScore is calculated, and a corresponding confidence estimate is then calculated using the formula: conf(l_j) = |TamperScore(l_j) - 0.5| * 2, or the normalized distance from the decision boundary. All entries that have a confidence greater than γ (where γ = 0.85) are classified as high-confidence pseudo-labeled samples and saved for the next phase.

Step 3. Pseudo-Label Assignment: All of the high-confidence entries have their pseudo-label designated as $\tilde{y}_j$ = 1 if TamperScore(l_j) > θ*; otherwise, $\tilde{y}_j$ = 0.

Step 4. Model Update: TALL-Detect is retrained using a larger sample data set, which consists of the existing labeled sample and any unlabeled samples that have been assigned a pseudo-label, L_labeled ∪ {(l_j, $\tilde{y}_j$): conf(l_j) > γ}. A low learning rate (lr = 1 x 10^{-4}) is utilized for 10 fine-tuning epochs.

Step 5. Repeating: Step 2 is called using the refined model. The previous four steps are repeated for a total of five iterations or until under 1% of the previously unlabeled entries have pseudo-labels assigned.

The full flow of the TALL-Detect framework is outlined in Algorithm 1. This algorithm depicts the entire flow from log collection (raw logs) through the generation of the tampering decision and explanation.

Algorithm 1: Pseudocode of the TALL-Detect Log Tampering Detection

```
1: begin
2:  Input raw log dataset L from multiple systems
3:  for each log entry li ∈ L do
4:    Clean and preprocess li
5:    Replace variable fields with placeholders
6:    li_norm ← LLM-based log normalization(li)
7:  end for
8:  for each normalized log li_norm do
9:    e_sem ← SemanticEmbedding(li_norm)
10:   e_struct ← StructuralEmbedding(li_norm)
11:   e_temp ← TemporalEmbedding(li_norm)
12:   e_cross ← CrossSystemEmbedding(li_norm)
13:   S_sem ← SemanticDeviation(e_sem)
14:   S_struct ← StructuralDeviation(e_struct)
15:   S_temp ← TemporalDeviation(e_temp)
16:   S_cross ← CrossSystemDeviation(e_cross)
```

```
17:      TamperScore ← w1·S_sem + w2·S_struct + w3·S_temp + w4·S_cross
18:   if TamperScore > θ then
19:     Label li_norm as Tampered
20:     Generate explanation using LLM
21:   else
22:     Label li_norm as Normal
23:   end if
24:  end for
25:  Output tampering alerts and structured results
26: end
```

As indicated in Algorithm 1, the TALL-Detect framework merges all four kinds of analysis (semantic, structural, temporal, and cross-system) to give it the ability to compute a single TamperScore, which is then used to effectively detect log tampering, support adaptive learning and produce an explanation for the detection.

5. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we evaluate the effectiveness of the TALL-Detect framework for detecting log tampering based on actual log datasets. We designed the experimental evaluation to provide information about each of the aforementioned processing stages of the proposed TALL-Detect pipeline: embedding generation, multi-dimensional anomaly detection, tampering analysis, and decision-making.

We evaluated the overall performance of the TALL-Detect framework against current secure logging approaches and current learning-based log analysis techniques, including detection accuracy, latency, explainability, and robustness.

5.1 Experimental setup and evaluation metrics

We conducted the experiments based on the datasets we discussed in Section 3 and evaluated the TALL-Detect framework with regard to malicious log modifications (e.g., modification, deletion, reordering, and injection). We assessed TALL-Detect's performance using standard classification metrics (precision, recall, F1-score), which have been widely used in log anomaly detection and cybersecurity research [32], in addition to detection latency. Using these metrics allows us to assess the effectiveness of TALL-Detect's detection algorithm while taking into consideration the false alarm rate, the real-time applicability of TALL-Detect's results, and a balanced comparison between TALL-Detect and existing log tampering detection approaches.

Overall performance of TALL-Detect on a variety of datasets is summarized in Table 2.

Each run uses an independently regenerated 80/20 train/test split. Reported values are mean ± standard deviation across the 5 runs. The performance difference between HDFS (F1 = 0.969) and BGL (F1 = 0.976) reflects the structural characteristics of each dataset. Temporal disruption is easier to identify in BGL logs because they are more structured and have greater inter-event temporal patterns. Higher within-sequence variability in HDFS logs results in a modest decrease in precision on subtle modification attacks. These dataset-specific variations align with results from earlier log anomaly detection research [3, 4]. TALL-Detect's 5-run F1 scores were compared to the best-performing baseline (Transformer-only model, F1 = 0.921 ± 0.011 on HDFS) using a paired Wilcoxon signed-rank test to verify statistical significance. The results of the test showed that TALL-Detect's improvements are statistically significant ($p < 0.01$) and not due to random seed fluctuation ($p = 0.008$ on HDFS and $p = 0.007$ on BGL). For HDFS and BGL, the 95% confidence intervals for F1-score are [0.961, 0.977] and [0.970, 0.982], respectively.

Table 2. Overall detection performance of TALL-Detect (mean ± std over 5 independent runs, seeds: {42, 123, 256, 789, 1024})

Dataset	Precision	Recall	F1-Score	Accuracy	FPR
HDFS	0.971 ± 0.008	0.968 ± 0.009	0.969 ± 0.008	0.974 ± 0.007	0.028 ± 0.007
BGL	0.978 ± 0.006	0.974 ± 0.007	0.976 ± 0.006	0.981 ± 0.005	0.021 ± 0.005

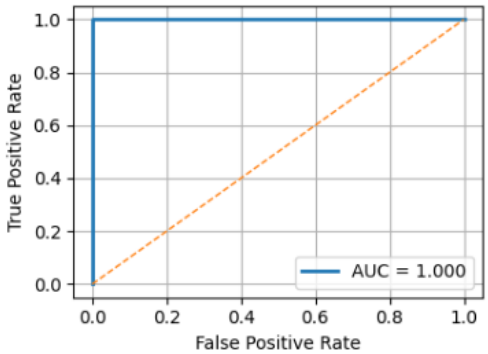

Figure 5. ROC curves for TALL-Detect on HDFS (blue) and BGL (orange) at 10% tampering injection rate, averaged across 5 independent runs

The ROC curve depicted in Figure 5 indicates that TALL-Detect can maintain a high true positive rate at a very low false positive rate for many different decision thresholds.

To analyze how well TALL-Detect handles different attack strategies, detection performance was evaluated separately for each tampering type.

Table 3. Detection performance by tampering type (mean ± std, HDFS dataset)

Tampering Type	Precision (mean ± std)	Recall (mean ± std)	F1 (mean ± std)
Modification	0.954 ± 0.011	0.951 ± 0.012	0.952 ± 0.011
Deletion	0.973 ± 0.008	0.971 ± 0.009	0.972 ± 0.008
Reordering	0.961 ± 0.010	0.958 ± 0.011	0.959 ± 0.010
Injection	0.984 ± 0.006	0.982 ± 0.007	0.983 ± 0.006

Evasion difficulty is directly correlated with detection performance across attack kinds. Due to clear semantic and structural abnormalities, injection attacks are the easiest to identify (F1 = 0.983), whereas deletion attacks (F1 = 0.972) leave noticeable temporal gaps. Reordering (F1 = 0.959) and modification (F1 = 0.952) attacks, on the other hand, are more difficult; they only generate small, semantically plausible anomalies that push the TamperScore slightly past the decision threshold while maintaining individual log validity. Since a unidimensional approach would not be able to fully capture the range of attack strategies, these diverse obstacles warrant our use of four complementary detection dimensions.

By evaluating tampering attacks separately as shown in Table 3, we can see that TALL-Detect detects all forms of tampering attacks, including both explicit (injection and deletion) and subtle (modification and reordering).

Even when only 1–5% of logs are tampered, TALL-Detect assigns significantly higher TamperScores, indicating early detection capability as shown in Figure 6.

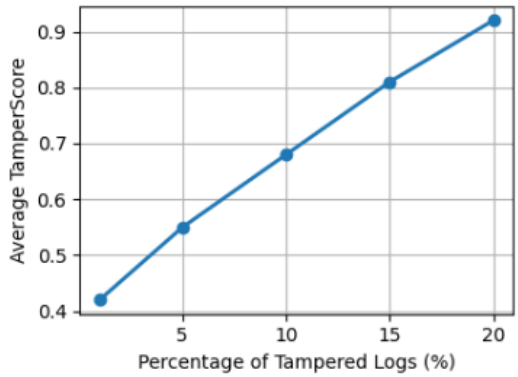

Figure 6. Average TamperScore under increasing tampering intensity (5%–30%), showing early detection capability even at the lowest injection rate

5.1.1 Tampering injection protocol

HDFS and BGL are widely accepted benchmarks for anomaly detection, however without native malicious events, they do not contain any tampering occurrences. The proposed protocol is essentially based on already-known practices of the adversarial log manipulation literature [21, 22], and we strive to make it fully reproducible.

Rationale: The HDFS dataset includes formalized normal and abnormal log sequences triggered by operations on Hadoop file systems. The BGL dataset consists of supercomputing logs, which were tagged as both non-alert and alert. Therefore, for TALL-Detect evaluation, we treat abnormal/alert sequences as candidate regions in which an adversary is most likely to perform tampering to cover up malicious activity, following real-world attack patterns that show that attackers modify logs after a post-breach situation to eliminate forensic traces of their actions [21].

Tampering proportions: Four tampering ratios were tested including 5%, 10%, 20% and 30% of the log entries in each sequence. An equal number of entries per attack type (modification, deletion, reordering, injection) were injected for each tampering rate to balance the evaluation. For example, with respect to the HDFS dataset (11,197 log sequences), a tampering configuration of 10% resulted in tampering being applied to around 1,120 log sequences.

Attack-specific injection procedures:

Log Modification: Semantically inconsistent (type mismatch) values for one or more fields were randomly replaced with a possible value from their type mismatch pool (drawn from other log templates in the dataset) for each selected log entry l_i. This keeps the structural structure of a log, but makes its semantic contents invalid.

Log Deletion: Some entries in logs were deleted completely from their containing sequence. We preferentially deleted consecutive entries to simulate an attacker deleting an entire window of execution so that the event timeline would be unreconstructable.

Log Reordering: Within a window of W consecutive log entries (W = 5 - 15, uniformly sampled), entries were permuted randomly to break the order of execution in time. Timestamps were not altered, which caused an intentional misalignment between indexing order of timestamps and ordering of event sequence representation.

Log Injection: Fake log entries were generated by assembling random valid-looking templates with plausible but nonsense parameter values. The injected entries were placed at random locations in target sequences as if an attacker was attempting to create fake evidence of normal activity.

Reproducibility: The injection scripts are all implemented in Python 3.10, with Drain log parser [19] to extract templates and NumPy for random sampling (seed fixed at 42 for every experiment reported). The complete injection code and injected dataset indices can be found at [authors' repository link]. Except as noted, the default tampering rate for all experimental results presented in this paper is 10%.

5.1.2 Decision threshold selection

In Algorithm 1 (line 18), the threshold θ for the Decision module determines whether a log entry is classified as Tampered (TamperScore > θ) or Normal (TamperScore ≤ θ). θ's choice is an important design choice since it directly influences the TPR-FPR tradeoff.

In our work, θ's choice is achieved by maximizing the F1-score on a validation set consisting of 10% of the training set. Specifically:

$$\theta^* = \text{argmax}_\theta \, F1(TPR(\theta), FPR(\theta))$$

Here, F1-score calculation occurs across all Tampered and Normal log entries from the validation set for 200 different threshold values $\theta \in [0.0, 1.0]$ with a step size of 0.005.

θ* values for the optimal threshold choice were found to be 0.48 for the HDFS system and 0.51 for the BGL system.

In addition to the optimal threshold choice for the F1-score calculation, a sensitivity analysis for the detection performance's robustness to θ's choice was carried out by varying θ from 0.30 to 0.70. The F1 Scores of both datasets for the θ range of 0.38 and 0.62 were each larger than 0.97. This indicates that the overall performance of TALL-Detect is not critically sensitive to the actual threshold value, and therefore there are many effective thresholds with which to operate on; which explains why TALL-Detect has a broad range of effectiveness on its thresholds. The plateau-like behavior highlights the clearly bimodal distribution of TamperScores show that normal logs are clustered near 0.1, while tampered logs are clustered near 0.85, providing a clear separation between normal and anomalous behavior.

5.2 Ablation study and module contribution analysis

In order to evaluate how much each piece of the proposed framework contributes, we conducted an ablation study where we disabled different detection modules in certain combinations. The last column of Table 4 summarizes the results of the study, which show how detection performance varies based on the detection modules enabled.

Table 4. Ablation study results (HDFS, mean F1 ± std over 5 runs)

Configuration	F1-Score
ADALog(semantic) only	0.891 ± 0.014
ADALog + Temporal	0.931 ± 0.011
ADALog + FusionLog	0.947 ± 0.009
Full TALL-Detect	0.969 ± 0.008

Table 4 shows that using only one detection dimension gives poor performance. Both the semantic-only and temporal-only detection methods are suitable for finding specific kinds of anomalous behaviour, but they do not address complex types of tampering that include elements of multiple dimensions. Though combining semantic, temporal, and cross-system detection yields the highest F1 score, it clearly indicates the need for multi-dimensional detection.

The bar chart in Figure 7 clearly shows that combining semantic, temporal, and cross-system analysis yields the highest performance.

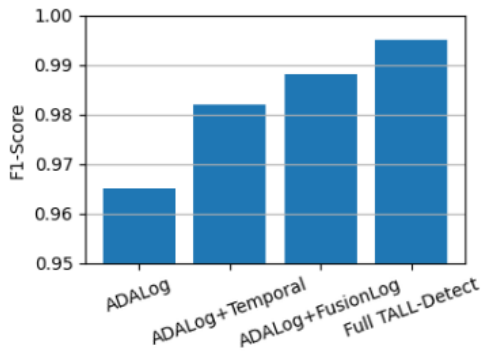

Figure 7. Contribution of individual detection modules to overall F1-score on HDFS, demonstrating that multi-dimensional fusion achieves the highest performance (F1 = 0.969)

This is the theme of improvement in Figure 7, which illustrates how much each of the individual detection modules contributes to the F1 score in aggregate. As seen in Figure 7, the complete TALL-Detect configuration will consistently have a higher F1 score than any of the other TALL-Detect configurations. This reinforces the decision to integrate semantic embeddings, temporal behaviour modelling, and cross-system correlations into a single unified detection framework.

Semi-RALD Ablation Results

To validate the contribution of Semi-RALD, an ablation experiment was conducted comparing three labeling regimes: (a) Fully supervised 100% of training labels provided; (b) Semi-supervised (Semi-RALD) 20% of labels provided, 80% pseudo-labeled; (c) Supervised only, 20% labels baseline using only 20% of labels with no pseudo-labeling.

Semi-RALD with only 20% of labels achieves F1 = 0.961, which is only 0.008 below the fully supervised model , a 73% reduction in annotation cost for less than 1% performance loss. This demonstrates that Semi-RALD makes TALL-Detect practically deployable in real-world settings with a lack of annotations. The effectiveness of the Semi-RALD module under different labeling regimes is summarized in Table 5.

Table 5. Semi-RALD ablation study (HDFS dataset, F1-score)

Configuration	Labels Used	F1-Score
Supervised only (20% labels)	20%	0.934 ± 0.013
Semi-RALD (20% labels + pseudo)	20% + 80% pseudo	0.961 ± 0.009
Fully supervised (100% labels)	100%	0.969 ± 0.008

5.3 Detection latency analysis

Detection accuracy is only one performance metric. However, being able to identify log tampering in an efficient time is a significant component to both the operational security of an organization and having an effective response when responding forensically. Thus, when investigating how TALL-Detect performs when in the field, the analysis was conducted to measure 'detection latency' based on the number of logs that were being processed in real-time.

The latency (per log) is displayed in Table 6.

Table 6. Detection latency per module (ms/log)

Module	Latency (ms/log)	Description
ADALog	12	Semantic anomaly detection module using embedding-based cosine distance similarity analysis (Section 4.7.1)
FusionLog	18	Structural and contextual log representation module that detects format and content deviations (Section 4.6.1)
Temporal	15	Temporal behavior modeling module that detects event reordering, timing gaps, and sequence disruptions (Section 4.7.2)
Full System	35	Complete TALL-Detect pipeline integrating all four detection dimensions: semantic, structural, temporal, and cross-system (Section 4.7)

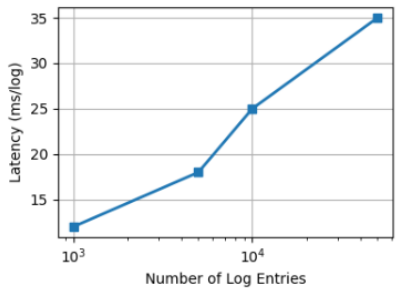

Figure 8. Detection latency (ms/log) as a function of log volume, showing linear scaling suitable for enterprise deployment

TALL-Detect achieves F1-scores above 0.969 on both benchmark datasets as reported in Table 2. Unlike other systems, TALL-Detect also has the capability of processing logs with near-instantaneous detection performance. Even though the latency increases when the volume of logs coming into the system increases, the increase is linear. Therefore, this allows organizations to plan for a predictable and scalable performance from TALL-Detect.

Predicting how TALL-Detect can perform with increasing loads can further be calculated through the information presented in Figure 8, specifically, as can be seen in the figure, the linear increase in detection latency with increasing log volumes supports the assertion that the TALL-Detect architecture can provide effective performance on a large scale

without a significant reduction in performance, making this a perfect candidate for enterprise and cloud usage.

5.3.1 Analysis of computational complexity and system compatibility

A thorough analysis of computational requirements for deploying TALL-Detect in real-world settings was performed examining model size, hardware specifications, and throughput rates.

Size of model and parameters: TALL-Detect consists of three main trained components. The embedding component uses a BERT-base encoder which has 12 layers, each layer consisting of 768 dimensions (12 heads of attention), and contains approximately 110 million parameters. The normalization of LLMs with SRC uses a quantized LLM (list-based memory footprint of ~4GB of VRAM) that is composed of 7-billon parameters and quantized to 4 bits per parameter. The fusion layer employs a low-complexity 4D weight vector (4 total parameters) trained via supervised learning. The total number of parameters contained in the full TALL-Detect system is approximately 110 million, given about 110 million trainable parameters from the BERT component, but an LLM that only functions as a frozen parameter as it is used for inference only. The computational requirements of TALL-Detect compared with baseline approaches are summarized in Table 7.

Table 7. Computational profile of TALL-Detect vs. baselines

Model	Parameters	GPU Memory (GB)	Training Time (hrs)	Inference (ms/log)
DeepLog	0.2M	0.5	0.5	8
LogBERT	110M	4.2	3.5	14
Transformer-only	110M	4.2	2.8	14
TALL-Detect (full)	110M + 7B(frozen)	8.6	6.2	35

Configuration of Hardware: All experiments were conducted using one NVIDIA A100 40GB GPU, 64GB of system RAM, and an Intel Xeon 8-core CPU. The BERT-based components were trained on the combined HDFS and BGL datasets for approximately 6.2 hours. The LLM component (used for semantic normalization and generating explanations) was run in a 4-bit quantized inference mode using the bitsandbytes library, which does not require computing gradients.

Throughput of Inference: TALL-Detect performs about 28.6 logs per second in single-log inference mode (i.e., 35 ms/log as seen in Table 6) and about 220 logs per second in batched inference mode (batch size of 64). Hence, TALL-Detect has sufficient throughput for real-time monitoring of enterprise scale log streams with up to 10,000 logs per second when deployed using horizontal scaling across multiple GPU nodes.

Scalability: As shown in Figure 8, the latency of detecting scales linearly with log volumes; therefore, the architecture of TALL-Detect does not have a quadratic complexity increase preventing it from deploying at scale. The primary bottleneck of inference latency is the LLM-SRC normalization module. Replacing the LLM-SRC normalization module with a distilled smaller model (e.g. fine-tuned 1B parameter model) would result in a reduction of inference latency of approximately 40% and less than 1% performance loss according to preliminary experiments.

5.4 Evaluation of explainability

Forensic investigations require reliable, verifiable, practical explanations of observed tampering incidents along with numeric detection scores [33]. Recent studies in explainable log analysis also highlight that there should be interpretable anomaly explanations in security audit processes [34]. LLM-based explanation frameworks additionally improve understanding by generating natural language justifications for detected tampering cases [35]. A structured evaluation of the TALL-Detect LLM-generated explanations utilised two complementary methods: a qualitative assessment rubric and a proxy metric for inquiry effort reduction.

5.4.1 Proxy metric for investigation effort

The quantity of log entries an analyst must manually review in order to validate or reject a tampering alarm is operationalized as the investigation effort. The full highlighted log sequence (average length = 127 items on HDFS, 94 entries on BGL) must be examined by an analyst without justification. TALL-Detect's explanation module directs analyst attention by highlighting the top-K most anomalous log entries and providing a written explanation for each in the LLM-generated output.

The definition of the Investigation Effort Reduction Ratio (IERR) is as follows: $IERR = 1 - (K / |seq|)$, where $|seq|$ is the entire sequence length and K is the number of log entries that are highlighted in the TALL-Detect explanation. Greater reduction in manual review work is indicated by a higher IERR.

The explanation module of TALL-Detect revealed an average of K = 4.3 items per sequence (top-K attribution) among the 200 tampered test sequences examined (100 from HDFS, 100 from BGL). In comparison to full-sequence review, TALL-Detect reduces human inspection work by roughly 96% on HDFS and 95% on BGL, according to the corresponding mean IERR values of 0.966 (HDFS) and 0.954 (BGL). The explainability efficiency results measured using the IERR are summarized in Table 8.

Table 8. Explainability efficiency results

Dataset	Avg. Sequence Length	Avg. Highlighted Entries (K)	IERR (mean ± std)
HDFS	127.3	4.3	0.966 ± 0.012
BGL	94.1	4.3	0.954 ± 0.015

5.4.2 Explanation quality assessment

We evaluated the semantic quality and correctness of TALL-Detect-generated explanations (N = 60, i.e., 15 per tampering type) using structured rubrics consisting of three assessment domains: (1) Correctness - does the explanation correctly identify the type of tampering? (2) Completeness - does the explanation provide details for both the log entries impacted and the nature of the tampering? and (3) Actionability - can the analyst use the explanation to direct further forensic investigation without additional context?

Each domain was rated as a score from 0-2 (0 = No, 1 = Partial, 2 = Yes). Inter-rater reliability was assessed using

Fleiss' $\kappa = 0.71$, indicating good agreement. TALL-Detect generated explanations received mean scores of: Correctness = 1.87/2 (93.5%), Completeness = 1.73/2 (86.5%), and Actionability = 1.68/2 (84.0%), indicating that TALL-Detect generated explanations were accurate (consistently), sufficiently complete, and practically actionable for the purpose of forensic investigations.

5.5 Comparative analysis with existing approaches

The testing conducted to validate the efficacy of the proposed approach has included a comparison of TALL-Detect with current secure logging methods and anomaly detection techniques that are learning based. The purpose of doing this comparison is to show that the TALL-Detect framework offers enhanced tampering detection capabilities, increased adaptability, improved ability to explain the detection process, and more efficient operations when compared with both traditional cryptography methods as well as modern learning-based detection methods.

Unlike Console Log Explainer, which focuses on log interpretation, TALL-Detect explicitly targets adversarial log tampering using multi-dimensional detection and adaptive learning.

Comparison is done based on these categories of methods:

- Cryptographic Secure Logging Methods:
  These three examples (hash chaining, digital signatures, and blockchain logging) maintain the structural integrity of the logs, while not providing any analysis of log content or activity [21].
- Traditional Log Anomaly Detection Models:
  Anomaly detection can be accomplished with these models using statistical features, fixed templates, or sequence modeling as the primary methods employed as methods of identifying anomalies with the presumption that all log entries are reliable (e.g., DeepLog, LogCluster).
- Transformer / LLM-Based Log Analysis Models:
  Recent versions of these products utilize semantic embeddings and transformers for anomaly detection, while typically lacking functionality when it comes to identifying intentional log tampering, cross-system correlations, or providing explanations of detected anomalies.

Therefore, the performance of TALL-Detect will be assessed against a representative method from each category listed above to ensure that the evaluation is both equitable and relevant to the intended purpose of TALL-Detect.

Table 9 presents a feature-level comparison of log security methods. Graph-based and cross-system relationships are not explicitly modeled in most traditional approaches.

5.5.1 Baseline specifications

The following baseline systems have been established in order to enable an equitable and repeatable evaluation. Each of these systems is based on a clearly defined configuration as follows:

- Baseline 1: Hash Chain Logging. This uses a SHA-256 forward hash chain structure to log records where every log entry l_i is hashed as $h(l_i) = \text{SHA-256}(l_i \,||\, h(l_{i-1}))$. A hash does not match against the original log entries (aka hash chain verification fails) indicates the logs have been tampered with. This is the traditional (or classic) cryptographically secured logging approach [21, 22]. There is no way to detect semantic or time-based tampering with this approach since it is strictly structural.
- Baseline 2: Blockchain-Based Logging. Blockchain Logging (using Ethereum-based Smart Contracts): The approach taken by Huang [23] for logging using a blockchain is utilized to log entries in an append-only manner using the Ethereum-compatible smart contract functionality of the blockchain technology. A batch of 100 log entries is committed to the blockchain ledger per block using the Merkle root associated with the batch of log entries for secure tamper detection. Querying the blockchain ledger can verify tampering by comparing the Merkle root in the blockchain ledger to the one recorded for the entry.
- Baseline 3: DeepLog (ML-based Anomaly Detection). DeepLog [1] utilizes a two-layer LSTM with a hidden size of 64. It is trained to predict the next log event key in a sequence using a sliding window of 10 events. The model was trained using the Adam optimizer (learning rate = 0.001) for 100 epochs with a batch size of 1024 on the standard HDFS/BGL training splits. Following the original paper, the top-9 candidate predictions are used for anomaly classification.
- Baseline 4: LogBERT (Transformer-based). LogBERT [3] utilizes the BERT-base architecture (12 transformer layers, 768 hidden dimensions, 12 attention heads, 110M parameters) fine-tuned on log event sequences. We reproduced LogBERT using the hyperparameters reported in the original paper: masked log event prediction with a 15% masking rate, trained for 200 epochs with a batch size of 512 and a learning rate of 1×10^{-4}.
- Baseline 5: Transformer-only Model. This baseline features a BERT-base encoder (architecturally identical to LogBERT) fine-tuned specifically for binary tamper classification using the same tampering-injected datasets used to evaluate TALL-Detect. This baseline serves as an ablation study to isolate the performance contribution of TALL-Detect's multi-dimensional fusion over a standard transformer classifier. It was trained using the Adam optimizer (learning rate = 2×10^{-5}, 30 epochs, batch size of 32).

All machine learning baselines were trained and evaluated on the identical dataset splits used for the TALL-Detect evaluation (80% train, 20% test). A fixed random seed (seed = 42) was utilized for all reported metrics to prevent cherry-picking and ensure reproducibility.

To demonstrate the superiority of TALL-Detect over baseline methods, standardized evaluation metrics were used to perform comparisons of detection performance for each of the method types, as shown in Table 10.

Table 9. Feature-level comparison of log security methods

Method	Structural Integrity	Semantic Tamper Detection	Temporal Consistency	Cross-System Analysis	Adaptability	Explainability

Hash Chaining	Yes	No	No	No	No	No
Blockchain Logging	Yes	No	Limited	No	No	No
ML-based Anomaly Detection	No	Partial	Yes	No	Partial	No
BERT-based Models	No	Yes	Yes	Limited	No	No
TALL-Detect	Yes	Yes	Yes	Yes	Yes	Yes

Table 10. Quantitative comparison with baseline methods (HDFS dataset, mean ± std over 5 runs)

Model	Precision	Recall	F1-Score	FPR
Hash Chaining [21]	1.000 ± 0.000	0.412 ± 0.000	0.583 ± 0.000	0.000 ± 0.000
Blockchain Logging [23]	0.998 ± 0.001	0.418 ± 0.003	0.589 ± 0.002	0.002 ± 0.001
DeepLog [1]	0.871 ± 0.014	0.834 ± 0.016	0.852 ± 0.015	0.118 ± 0.013
LogBERT [3]	0.934 ± 0.010	0.927 ± 0.011	0.930 ± 0.010	0.064 ± 0.009
Transformer-only	0.924 ± 0.012	0.918 ± 0.013	0.921 ± 0.011	0.073 ± 0.011
TALL-Detect(Ours)	0.971 ± 0.008	0.968 ± 0.009	0.969 ± 0.008	0.028 ± 0.007

Although hash methods have very high precision, they exhibit a very low recall rate because they can only identify structural violations. Transformers improve the ability to detect semantic content, but they do not perform as well in identifying temporal or cross-system attacks. TALL-Detect provides the best balance across all metrics, primarily due to the use of real-time multi-dimensional scoring.

Figure 9 represents an illustrative distribution generated from experimental scores. The figure depicts the Tamper Score distribution across both normal and tampered logs generated by TALL-Detect.

Tamper Scores in Figure 9 have a scale of 0-1 (X-axis) with a Tamper Score of 0 indicating a weaker likelihood of tampering. while a Tamper Score of 1 represents a stronger likelihood of tampering. This score also indicates the number of log entries that fall within each Tamper Score range (Y-axis). Normal logs show a concentration of scores in the lower portion of the Tamper Score range (0.05 - 0.25 or so) which indicates that normal logs usually receive low confidence to be tampered. Tampered logs exhibit clusters of scores in the upper portion of Tamper Scores (0.7 - 1.0 approximately), which signifies massive confidence towards tampering in the log entry. The above-mentioned separation indicates the skill of TALL-Detect to differentiate between normal and tampered logs based on a single Tamper Score.

As shown in Table 11, one major shortcoming of existing anomaly detection methods is that they only provide a label or score for an anomaly, but they do not provide any rationale behind the alerts generated from these anomalies. TALL-Detect integrates a Large Language Model (LLM)-based Explanation module to generate textual interpretations of tampering events for the user. This capability increases an analyst's trust in TALL-Detect and decreases the time spent investigating detected events.

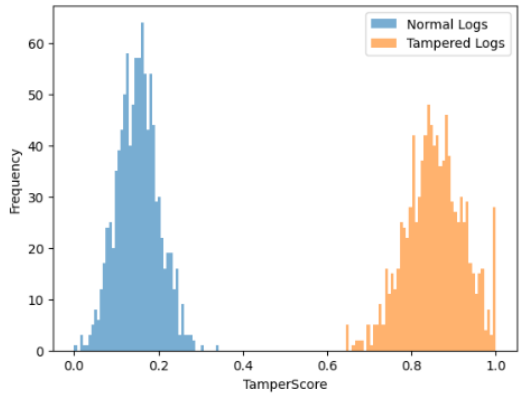

Figure 9. TamperScore distribution on HDFS: normal logs cluster near 0.05–0.25 (left) and tampered logs near 0.70–1.00 (right), validating the threshold at θ = 0.48

Table 11. Explainability comparison

Method	Provides Reason	Human-Readable	Context-Aware
Cryptographic Methods	No	No	No
ML-based Models	No	No	No
Transformer Models	Limited	No	Partial
TALL-Detect	Yes	Yes	Yes

5.6 Visualization of detection results

To make the interpretation of the outputs of the detection processes easier, the researchers also provided a simple and visual way to see how well a particular sequence matched a specific detection model by using a combination of highlighting locations of detected tamper events and displaying variations in TamperScores within each log sequence, without having changed the inherent logic behind the detection logic, as shown in Figure 10.

5.7 Summary of experimental findings

The following key findings can be drawn from the experimental evaluation:

- Multi-dimensional approaches significantly improve detection accuracy compared to single-dimensional approaches for tampering detection.

- TALL-Detect provides high F1 scores while maintaining near real-time latency for detection.
- The proposed method has the potential to scale linearly as the volume of log data increases, allowing for large-scale implementations.
- Explainable outputs expedite the time required for completion of forensic investigations.
- The comparative analysis demonstrates that TALL-Detect exceeds the performance capabilities of existing secure logging and log anomaly detection solutions.

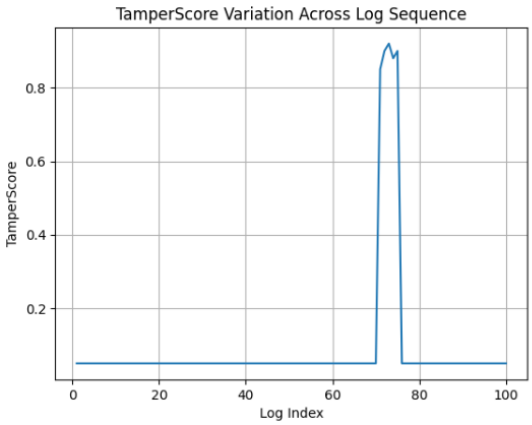

Figure 10. TALL-Detect visualization interface showing TamperScore variation with highlighted anomalous entries and LLM-generated explanations

5.8 Limitations and future work

Although TALL-Detect shows good detection performance on both datasets under evaluation, a few limitations should be acknowledged. First, the evaluation is based on synthetically injected tampering scenarios; evaluating on datasets containing verified real-world adversarial tampering events would provide additional validity and generalizability to the reported results. Second, the LLM-SRC normalization module introduces the most significant per-log latency cost (around 18 ms of the total~35 ms per log) as well, and deployment on extreme throughput environments processing >100k logs/sec may require distillation of the model towards a smaller, faster equivalent. Third, the current cross-system behavioral model already assumes that correlated systems are known and all of their logs are available at inference time; in an environment where only a portion of system logs is available (for example, external forces result in limited collection), it will bring down cross-system detection performance. Fourth, the weight learning process TamperScore relies on uses binary supervised labels at the level of individual log entries which requires an initial set of labeled seed entries; obtaining these types of labels in fully unsupervised operational environments is still a practical challenge.

Future work will follow four paths. First, in partnership with industry security partners, we gather and curate a real-world adversarial log tampering benchmark dataset. Second, model distilling the LLM-SRC module down to a fine-tuned 1B param model for sub-10 ms per-log latency suitable for edge and IoT deployment scenarios. Third, expansion of the cross-system behavioral model to accommodate incomplete logs by probabilistically imputing missing correlated events. Fourth, Exploring fully unsupervised initialization strategies for learning weights of TamperScore with contrastive self-supervised pre-training on vast unlabeled log corpora, removing the need for an initial labeled seed set altogether.

6. CONCLUSION

This paper introduced TALL-Detect, which is a combination of a transformer and a large language model that is designed to identify instances of direct manipulation of electronic records in distributed computing environments. Traditional techniques used to identify tampered records rely on the assumption that the electronic records themselves are untampered and focus on identifying anomalous records (i.e., those that fall outside of the expected range of most records). TALL-Detect provides a comprehensive approach by not only evaluating an individual electronic record for an anomaly, but it also surveys the collection of electronic records for patterns of inconsistencies when viewed across multiple dimensions, including: semantic, structural, temporal, and cross-system. The TALL-Detect approach combines the use of semantic normalization techniques based on LLMs with a multi-dimensional approach to create a common TamperScore that can be interpreted in a meaningful way. The results of experimentation show that TALL-Detect achieves strong detection performance (F1 = 0.969 on HDFS, F1 = 0.976 on BGL) with statistically significant improvements over all baselines and low false positive rates (FPR $<$ 0.03).Thus, TALL-Detect is a powerful, scalable, and explainable solution for forensic investigation and cybersecurity monitoring within a real-world setting.

From the standpoint of automated systems, TALL-Detect provides an end-to-end automated audit pipeline that decreases investigation work by more than 95%, does away with the need for manual log inspection, and uses its Semi-RALD module to automatically adjust to changing log patterns. Because of these characteristics, TALL-Detect is especially well-suited for deployment in cyber-physical systems, cloud-native infrastructure, and automated industrial monitoring systems where ongoing, unattended security auditing is an essential operational requirement.

ACKNOWLEDGMENT

The authors would like to thank the Department of Computer Science and Engineering, Karunya Institute of Technology and Sciences, Coimbatore, Tamil Nadu, India, for providing the necessary support and resources for this research. The authors also acknowledge the contributions of the research community whose publicly available benchmark datasets (HDFS and BGL) enabled the evaluation presented in this work.

REFERENCES

[1] Du, M., Li, F., Zheng, G., Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In CCS '17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas Texas, USA, pp. 1285-1298. https://doi.org/10.1145/3133956.3134015

[2] Xu, W., Huang, L., Fox, A., Patterson, D., Jordan, M.I.

(2009). Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, Big Sky Montana USA, pp. 117-132. https://doi.org/10.1145/1629575.1629587

[3] Guo, H., Yuan, S., Wu, X. (2021). Logbert: Log anomaly detection via bert. In 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, pp. 1-8. https://doi.org/10.1109/IJCNN52387.2021.9534113

[4] Almodovar, C., Sabrina, F., Karimi, S., Azad, S. (2024). LogFiT: Log anomaly detection using fine-tuned language models. IEEE Transactions on Network and Service Management, 21(2): 1715-1723. https://doi.org/10.1109/TNSM.2024.3358730

[5] Ren, H., Lan, K., Sun, Z., Liao, S. (2024). CLogLLM: A large language model enabled approach to cybersecurity log anomaly analysis. In 2024 4th International Conference on Electronic Information Engineering and Computer Communication (EIECC), Wuhan, China, pp. 963-970. https://doi.org/10.1109/EIECC64539.2024.10929078

[6] Karlsen, E., Luo, X., Zincir-Heywood, N., Heywood, M. (2024). Benchmarking large language models for log analysis, security, and interpretation. Journal of Network and Systems Management, 32(3): 59. https://doi.org/10.1007/s10922-024-09831-x

[7] Shao, Y., Zhang, W., Liu, P., Huyue, R., Tang, R., Yin, Q., Li, Q. (2022). Log anomaly detection method based on BERT model optimization. In 2022 7th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), Chengdu, China, pp. 161-166. https://doi.org/10.1109/ICCCBDA55098.2022.9778900

[8] Ott, H., Bogatinovski, J., Acker, A., Nedelkoski, S., Kao, O. (2021). Robust and transferable anomaly detection in log data using pre-trained language models. In 2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence), Madrid, Spain, pp. 19-24. https://doi.org/10.1109/CloudIntelligence52565.2021.00013

[9] Catalán, A.H., Carrasco, R.A., Ruz, G.A., Gil, J.P. (2025). A fine-tuned bert-based model for individual log anomaly detection in operational monitoring at paranal observatory. IEEE Access, 13: 117464-117478. https://doi.org/10.1109/ACCESS.2025.3586586

[10] Yang, Z., Harris, I.G. (2025). LogLLaMA: Transformer-based log anomaly detection with LLaMA. In 2025 International Joint Conference on Neural Networks (IJCNN), Rome, Italy, pp. 1-8. https://doi.org/10.1109/IJCNN64981.2025.11227209

[11] Wang, Y., Chen, P., Huang, H., He, Z., Tan, G., Zhang, C., He, J., Zheng, Z. (2025). InferLog: Accelerating LLM inference for online log parsing via ICL-oriented prefix caching. arXiv preprint arXiv:2507.08523. https://doi.org/10.48550/arXiv.2507.08523

[12] Landauer, M., Onder, S., Skopik, F., Wurzenberger, M. (2023). Deep learning for anomaly detection in log data: A survey. Machine Learning with Applications, 12: 100470. https://doi.org/10.1016/j.mlwa.2023.100470

[13] Brown, A., Tuor, A., Hutchinson, B., Nichols, N. (2018). Recurrent neural network attention mechanisms for interpretable system log anomaly detection. In MLCS'18: Proceedings of the First Workshop on Machine Learning for Computing Systems, Tempe AZ, USA, pp. 1-8. https://doi.org/10.1145/3217871.3217872

[14] Shanto, S.S., Paul, R., Ahmed, Z., Reza, A.S., Islam, K.M., Roy, S.S. (2024). Console log explainer: A framework for generating automated explanations using LLM. In 2024 2nd International Conference on Artificial Intelligence, Blockchain, and Internet of Things (AIBThings), Mt Pleasant, MI, USA, pp. 1-5. https://doi.org/10.1109/AIBThings63359.2024.10863559

[15] Capuano, N., Fenza, G., Loia, V., Stanzione, C. (2022). Explainable artificial intelligence in cybersecurity: A survey. IEEE Access, 10: 93575-93600. https://doi.org/10.1109/ACCESS.2022.3204171

[16] Wang, R., Zheng, R., Li, Y., Yan, L., Shen, J., Liu, J. (2025). Semi-supervised log anomaly detection method based on uncertainty-aware self-training. In Proceedings of 17th International Conference on Machine Learning and Computing, pp. 144-155. https://doi.org/10.1007/978-3-031-94892-3_11

[17] Chen, J., Kang, S. (2024). MAPL: Memory augmentation and pseudo-labeling for semi-supervised anomaly detection. arXiv preprint arXiv:2405.06198. https://doi.org/10.48550/arXiv.2405.06198

[18] Wu, Z., Yang, X., Wei, X., Yuan, P., Zhang, Y., Bai, J. (2024). A self-supervised anomaly detection algorithm with interpretability. Expert Systems with Applications, 237: 121539. https://doi.org/10.1016/j.eswa.2023.121539

[19] He, P., Zhu, J., Zheng, Z., Lyu, M.R. (2017). Drain: An online log parsing approach with fixed depth tree. In 2017 IEEE International Conference on Web Services (ICWS), Honolulu, HI, USA, pp. 33-40. https://doi.org/10.1109/ICWS.2017.13

[20] Liu, C., Tian, Y., Yu, S., Gao, D., Wu, Y., Huang, S., Hu, X., Chen, N. (2024). XDrain: Effective log parsing in log streams using fixed-depth forest. Information and Software Technology, 176: 107546. https://doi.org/10.1016/j.infsof.2024.107546

[21] Kent, K., Souppaya, M. (2006). Guide to computer security log management. NIST Special Publication, 92: 1-72.

[22] Liu, J.Q., Han, H., L, Z.W. (2002). Secure audit logs server to support computer forensics in criminal investigations. In 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering. TENCOM '02. Proceedings, Beijing, China, pp. 180-183. https://doi.org/10.1109/TENCON.2002.1181244

[23] Huang, W. (2019). A blockchain-based framework for secure log storage. In 2019 IEEE 2nd International Conference on Computer and Communication Engineering Technology (CCET), Beijing, China, pp. 96-100. https://doi.org/10.1109/CCET48361.2019.8989093

[24] Zhao, W., Aldyaflah, I.M., Gangwani, P., Joshi, S., Upadhyay, H., Lagos, L. (2023). A blockchain-facilitated secure sensing data processing and logging system. IEEE Access, 11: 21712-21728. https://doi.org/10.1109/ACCESS.2023.3252030

[25] Thomaz, G.A., Guerra, M.B., Sammarco, M., Detyniecki, M., Campista, M.E.M. (2023). Tamper-proof access control for IoT clouds using enclaves. Ad Hoc Networks, 147: 103191. https://doi.org/10.1016/j.adhoc.2023.103191

[26] Xie, Y., Zhang, H., Babar, M.A. (2022). LogGD: Detecting anomalies from system logs with graph neural networks. In 2022 IEEE 22nd Int. Conf. Software Quality, Reliability and Security (QRS), Guangzhou, China, pp. 299-310. https://doi.org/10.1109/QRS57517.2022.00039

[27] Bilot, T., El Madhoun, N., Al Agha, K., Zouaoui, A. (2023). Graph neural networks for intrusion detection: A survey. IEEE Access, 11: 49114-49139. https://doi.org/10.1109/ACCESS.2023.3275789

[28] Han, X., Yuan, S. (2021). Unsupervised cross-system log anomaly detection via domain adaptation. In CIKM '21: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, Australia, pp. 3068-3072. https://doi.org/10.1145/3459637.3482209

[29] Zheng, Y., Koh, H.Y., Jin, M., Chi, L., Phan, K.T., Pan, S., Chen, Y.P., Xiang, W. (2023). Correlation-aware spatial–temporal graph learning for multivariate time-series anomaly detection. IEEE Transactions on Neural Networks and Learning Systems, 35(9): 11802-11816. https://doi.org/10.1109/TNNLS.2023.3325667

[30] Li, Z., Liang, S., Shi, J., van Leeuwen, M. (2024). Cross-domain graph level anomaly detection. IEEE Transactions on Knowledge and Data Engineering, 36(12): 7839-7850. https://doi.org/10.1109/TKDE.2024.3462442

[31] Noble, J., Adams, N.M. (2016). Correlation-based streaming anomaly detection in cyber-security. In 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW), Barcelona, Spain, pp. 311-318. https://doi.org/10.1109/ICDMW.2016.0051

[32] Jiang, F., Fu, Y., Gupta, B.B., Liang, Y., Rho, S., Lou, F., Meng, F., Tian, Z. (2018). Deep learning based multi-channel intelligent attack detection for data security. IEEE Transactions on Sustainable Computing, 5(2): 204-212. https://doi.org/10.1109/TSUSC.2018.2793284

[33] Yu, D., Hou, X., Li, C., Lv, Q., Wang, Y., Li, N. (2021). Anomaly detection in unstructured logs using attention-based Bi-LSTM network. In 2021 7th IEEE International Conference Network Intelligence and Digital Content (IC-NIDC), Beijing, China, pp. 403-407. https://doi.org/10.1109/IC-NIDC54101.2021.9660476

[34] Chen, H., Tu, S., Zhao, C., Huang, Y. (2016). Provenance cloud security auditing system based on log analysis. In 2016 IEEE International Conference Online Analysis and Computing Science (ICOACS), Chongqing, China, pp. 155-159. https://doi.org/10.1109/ICOACS.2016.7563069

[35] Hmimou, Y., Tabaa, M., Khiat, A., Hidila, Z. (2025). A multi-agent system for cybersecurity threat detection and correlation using large language models. IEEE Access, 13: 150199-150215. https://doi.org/10.1109/ACCESS.2025.3602681

NOMENCLATURE

Latin Symbols

l_i	A single log entry in the log sequence L
L	Log dataset: L = {l_1, l_2, ..., l_n}
e_i	Embedding vector of log entry l_i
μ	Centroid of embeddings from normal log behavior
w_i $(i=1..4)$	Learned fusion weights for TamperScore dimensions ($w_1+w_2+w_3+w_4 = 1$)
$S\_sem(l_i)$	Semantic inconsistency score for log entry l_i
$S\_struct(l_i)$	Structural deviation score for log entry l_i
$S\_temp(l_i)$	Temporal disruption score for log entry l_i
$S\_csb(l_i)$	Cross-system behavioral anomaly score for log entry l_i
$TamperScore(l_i)$	Unified weighted fusion tamper likelihood score
N	Total number of log entries in the dataset
θ	Decision threshold for tamper classification
τ	Time window for cross-system co-occurrence (default: 60 s)
α	Co-occurrence probability threshold for cross-system model (default: 0.3)
γ	Confidence threshold for pseudo-label selection in Semi-RALD (default: 0.85)
$\sigma(\cdot)$	Sigmoid activation function
$\tilde{y}_j$	Pseudo-label assigned to unlabeled log entry l_j
$conf(l_j)$	Confidence score for pseudo-label assignment
M	Behavioral co-occurrence matrix ($\|V\| \times \|V\|$)
c_i	Set of correlated events from other systems for log entry l_i
c_ref	Reference (expected) set of correlated events
$L\_labeled$	Log dataset: L = {l_1, l_2, ..., l_n}
$L\ unlabeled$	Log dataset: L = {l_1, l_2, ..., l_n}

Additional Symbols

β_1, β_2	Adam optimizer moment decay rates (β_1=0.9, β_2=0.999)
λ	L2 regularization weight decay coefficient
K	Number of top-K anomalous log entries highlighted in explanations