Bridging the Gap: LLM-Powered Transfer Learning for Log Anomaly Detection in New Software Systems

Yicheng Sui*, Xiaotian Wang*, Tianyu Cui*, Tong Xiao†, Chenghao He*,
Shenglin Zhang+*, Yuzhi Zhang*, Xiao Yang‡, Yongqian Sun*, Dan Pei†
*Nankai University, Tianjin, China †Tsinghua University, Beijing, China
‡China Mobile Communications Corporation, China
suiyicheng@mail.nankai.edu.cn, 2120240811@mail.nankai.edu.cn, 1120220298@mail.nankai.edu.cn,
xiaotong@tsinghua.edu.cn, 2011751@mail.nankai.edu.cn, zhangsl@nankai.edu.cn, zyz@nankai.edu.cn,
yangxiaoyjy@chinamobile.com, sunyongqian@nankai.edu.cn, peidan@tsinghua.edu.cn

Abstract—**For large IT companies, maintaining numerous software systems presents considerable complexity. Logs are invaluable for depicting the state of systems, making log-based anomaly detection crucial for ensuring system reliability. Existing methods require extensive log data for training, hindering their rapid deployment for new systems. Cross-system log anomaly detection methods attempt to transfer knowledge from mature systems to new ones but often struggle with syntax differences and system-specific knowledge, which hinders their effectiveness. To address these issues, this paper proposes LogSynergy, a novel transfer learning-based log anomaly detection framework. LogSynergy employs (1) LLM-based event interpretation (LEI) to standardize log syntax across different systems, and (2) system-unified feature extraction (SUFE) to disentangle system-specific features from system-unified features. These bridge the gap among different systems and enhance LogSynergy's generalizability. LogSynergy has been deployed in the production environment of a top-tier global Internet Service Provider (ISP), where it was evaluated on three real-world datasets. Additionally, we conducted evaluations on three public datasets. The results demonstrate that LogSynergy significantly outperforms existing methods. It achieves F1-scores over 89% on the real-world datasets and over 83% on the public datasets, using only 5000 labeled log sequences from the new system. These results underscore LogSynergy's effectiveness in rapidly deploying anomaly detection models for new systems. The code of LogSynergy has been open-sourced at https://anonymous.4open.science/r/LogSynergy-320D/.**

Index Terms—**log anomaly detection, transfer learning, feature disentanglement**

I. Introduction

Large-scale software systems are being rapidly developed and implemented, presenting significant challenges for Internet Service Providers (ISPs) in maintaining reliable and high-performance services. Within ISPs, Cloud Data Management Systems (CDMSs) are critical for managing vast amounts of data across distributed cloud environments, serving as the backbone for essential services. The complexity of these systems necessitates efficient monitoring to ensure service continuity, and it becomes increasingly important to identify anomalies quickly and accurately to prevent disruptions or performance degradation. Moreover, the complexity of software systems makes it imperative to identify anomalies promptly and accurately to prevent service disruptions or performance degradation. Against this backdrop, log data plays an essential role, as it records software components' status, activities, and significant events and offers comprehensive insights into system operations. Thus, log data emerges as a valuable resource for anomaly detection. Log-based anomaly detection can facilitate the rapid identification of software system issues in CDMS, enabling ISPs to avert severe consequences of failures.

For the CDMS in large ISPs, it is common to deploy or update new software systems to accommodate the evolving business needs of the ISP. As a result, it is crucial to quickly deploy anomaly detection methods tailored for these newly deployed systems to ensure their reliability and stability from the outset. However, obtaining sufficient data and labels, especially for anomalous logs, is a challenging task [1], [2]. To mitigate this, cross-system log anomaly detection methods, utilizing transfer learning [2]–[4] or meta-learning [1], have been proposed to leverage knowledge from similar mature systems. By transferring knowledge from similar mature systems, the cross-system methods reduce the reliance on sufficient labeled data from new systems. Fortunately, in large ISPs, where many systems with similar functionalities are deployed, identifying such similar systems is relatively easy, making cross-system anomaly detection an effective solution for rapidly deploying anomaly detection models.

However, existing cross-system log anomaly detection methods typically utilize the syntax similarity between log messages from mature and new systems. Our observations from real-world log messages reveal that similar anomalous events in different systems may exhibit significant syntax differences. For example, Table I lists four log messages from two different systems covering two anomalous events. These logs originate from two different systems, resulting in significant differences in syntax. Specifically, the first two

+ Corresponding Author.

TABLE I: Log messages that represent similar anomalies from different systems. (Words marked in red indicate anomalies)

Anomalous Event	Dataset	Log Message
Network Interruption	Spirit [5]	Connection refused (111) in open_demux, open_demux: connect 172.30.72.31:33404
	BGL [5]	ciod: failed to read message prefix on control stream (CioStream socket to 172.16.96.116:33399)
Parity Error	Spirit [5]	GM: LANAI[0]: PANIC: mcp/gm_parity.c:124 : parity__int():firmware
	BGL [5]	machine check interrupt (bit=0x10): L2 dcache unit read return parity error

log messages are both about network interruptions. However, the first log message details the issue occurring during the "open_demux" operation. In contrast, the second log message records the component involved in the issue. Similarly, the third and fourth log messages document parity errors while differ significantly in detail and focus. These illustrate the substantial syntax differences in log messages from different systems, even when they indicate the same anomalous event. This difference hinders knowledge from source systems to be effectively transferred to target systems, impacting the effectiveness of transfer learning. Consequently, existing transfer learning-based methods usually still require sufficient logs from the new system, severely limiting their application in new systems.

Moreover, there is another issue when transferring knowledge from mature systems to new ones. Because log messages from mature systems are far more than those from new systems, the transferred knowledge would include much system-specific knowledge from the mature system which is not applicable to the target system. So it is crucial to filter out the system-specific knowledge to transfer the knowledge effectively.

In summary, the significant syntax differences among different systems and the system-specific knowledge from mature systems create a gap that hinders effective knowledge transfer. To address these issues, we propose LogSynergy, a novel transfer learning-based log anomaly detection method that is designed to bridge this gap and enable rapid deployment for new software systems. LogSynergy first unifies the log syntax of different systems using a large language model (LLM)-based event interpretation method. It then disentangles system-specific features and system-unified features and only leverages the system-unified features to transfer knowledge from mature systems to new systems. The main contributions of this paper can be summarized as follows:

1. LLM-based Event Interpretation (LEI): To reduce the log syntax differences in different systems, we propose an LLM-based event interpretation (LEI) method. By leveraging the semantic understanding capability of large language models (LLMs) [6], LEI standardizes the syntax of log messages from different systems, which can enhance the model's generalization capability.

2. System-Unified Feature Extraction (SUFE): To filter out the system-specific knowledge from mature systems, we propose a system-unified feature extraction (SUFE) method. It disentangles system-specific features from system-unified features by minimizing the mutual information [7] among different systems and filters out system-specific knowledge by filtering out system-specific features.

3. Extensive Evaluation on Real-World and Public Datasets: We validate LogSynergy's performance through experiments on three log datasets from the CDMS of a top-tier global ISP and three public log datasets. The evaluation underscores LogSynergy's effectiveness in handling diverse, real-world log data and establishes its advantages over existing log anomaly detection methods in log-based anomaly detection.

II. Preliminaries and Related Work

A. Preliminaries

Transfer learning is used to improve model performance in a target domain by leveraging knowledge acquired from a related source domain, particularly in scenarios where labeled data in the target domain is scarce [8]. Formally, given a source domain-task pair (D_S, T_S) and a target domain-task pair (D_T, T_T), transfer learning aims to improve f_T in the target domain by leveraging knowledge from a well-annotated source domain, even when distributions or feature spaces differ. It focuses on instance weighting, feature transformation, and model adaptation to bridge domain discrepancies, mitigating performance degradation and enhancing generalization [9].

Domain adaptation is a subfield of transfer learning that aims to address performance degradation caused by domain shift, minimizing distributional discrepancies between source and target domains. This allows models trained on labeled source data to generalize effectively to the target domain [10]. Common approaches in domain adaptation include distribution alignment methods, such as Maximum Mean Discrepancy (MMD) [11], and adversarial learning techniques like Dynamic Adversarial Adaptation Network (DAAN) [12]. In LogSynergy, we use the DAAN for domain adaptation. DAAN dynamically evaluates the relative importance of marginal and conditional distributions, improving the alignment between the feature distributions of the source and target domains. This method enhances the model's ability to generalize across different domains by reducing domain-specific variations.

B. Related Work

In the field of log anomaly detection, various methods have been proposed to address the challenges of detecting anomalies in logs from new systems. As listed in Table II, these methods can be categorized into six categories. Each category has its strengths in specific scenarios, but all face certain limitations when applied to new systems, especially when the new system's data is scarce.

The existing cross-system methods [1]–[4] often rely on large amounts of labeled or normal data, which is difficult to

TABLE II: Related work on log anomaly detection and their limitations in new systems. (Works from the database systems community in bold)

Method Category	Methods	Main Characteristics	Limitations in New System
Supervised Cross-System	MetaLog [1], LogTransfer [2]	Require sufficient labeled data for transfer/meta-learning.	Limited labeled data in new systems reduces effectiveness.
Unsupervised Cross-System	LogTAD [4], TransferLog [3]	Rely on sufficient normal data or pseudo-labels for transfer learning.	Limited data in new systems reduces transfer learning effectiveness.
Unsupervised Single-System	DeepLog [13], LogAnomaly [14], **Pluto** [15], LogFlash [16], **ADOps** [17], **RT-Log** [18]	Require sufficient normal data to learn normal patterns by unsupervised learning. No cross-system adaptability.	Limited data in new systems makes it difficult to capture normal patterns.
Semi-Supervised Single-System	PLELog [19], BTCNLog [20], AFALog [21], **SpikeLog** [22]	Require sufficient labeled and unlabeled data for semi-supervised learning. Lack of cross-system adaptability.	Few data from new systems limits the performance. Moreover, the anomaly detection knowledge of mature systems cannot be transferred to new systems.
Supervised Single-System	LogRobust [23], NeuralLog [24], OneLog [25], **MoniLog** [26]	Require sufficient labeled data for supervised learning. Lack of cross-system adaptability.	
LLM-Based	LogGPT [27], **PreLog** [28]	Leverage pre-trained LLMs for semantic comprehension and anomaly detection.	Limited knowledge from LLMs and few new systems' data hinder LogGPT's performance. The scarcity of new system data limits prompt-tuning effectiveness of PreLog.

obtain in new systems, limiting their effectiveness in transfer learning. TransferLog [3] and LogTAD [4] use domain adaptation techniques but still require substantial normal data to maintain model performance. LogTransfer [2] and MetaLog [1] employ transfer learning and meta-learning strategies, respectively, to adapt models across systems, but they rely on sufficient labeled data in the new system.

Similarly, single-system methods perform well within a single system but cannot transfer anomaly detection knowledge across systems, making them unsuitable for detecting anomalies in new systems. Research in this area is extensive and can be broadly categorized into unsupervised, semi-supervised, and supervised methods. Unsupervised methods, such as DeepLog [13], LogAnomaly [14], Pluto [15], LogFlash [16], ADOps [17], and RT-Log [18], rely on sufficient normal data, while semi-supervised methods, including PLELog [19], BTCNLog [20], AFALog [21], and SpikeLog [22], use a small amount of labeled data for guidance. Supervised methods, such as LogRobust [23], NeuralLog [24], OneLog [25], and MoniLog [26], require fully labeled datasets. Despite their strengths in single-system settings, these methods are limited when data is scarce in new system scenarios and cannot be directly applied to cross-system anomaly detection.

LLM-based methods enhance anomaly detection through advanced semantic and contextual understanding, leveraging pre-trained embeddings for generalization across datasets with minimal retraining. These LLM-based methods have demonstrated strong performance in processing time-series and timestamped log data [29]. While approaches like LogGPT [27] and PreLog [28] enable zero-shot or few-shot detection, they struggle to effectively transfer anomaly detection knowledge from mature systems to new systems. Moreover, the limited availability of labeled data in new systems further hinders their effectiveness, making them less practical in new software systems.

LogSynergy aims to bridge the gap between different systems to enhance transfer learning, enabling effective learning with little data from the new systems and overcoming the limitations of existing methods that struggle with insufficient data of the new system.

III. Design of LogSynergy

A. Overview

Fig.1 details the framework of LogSynergy. In the offline phase, LogSynergy initially converts unstructured raw logs into structured log events and parameters. Then, it splits continuous log events into event sequences. Subsequently, it analyzes the log events using LLM-based event interpretation (LEI) to obtain event interpretations and converts these event interpretations to event embeddings. The event embeddings and log event sequences from both mature systems (*i.e.*, source systems) and the new system (*i.e.*, target system) are then input into a classification model based on the Transformer Encoder [30] to train an anomaly detection model offline. LogSynergy employs system-unified feature extraction (SUFE) during training to filter out system-specific features. When the target system generates log messages in the online phase, LogSynergy extracts their features by the same method used in the offline phase and evaluates them against the trained model to determine whether they are anomalies.

In the rest of this section, we will introduce the pre-processing phase in §III-B, followed by event representation in §III-C, with a focus on LEI. Then, we will elaborate on the transfer learning in §III-D and emphasize SUFE in §III-D2. Finally, we will describe the online detection phase in §III-E.

B. Pre-processing

The pre-processing phase is the first step in the LogSynergy framework. In this phase, unstructured raw logs from various source and target systems are converted into a structured format suitable for further analysis.

First, raw logs are collected from various systems. These logs are unstructured, comprising log messages, timestamps, and other relevant information generated during system operations. Next, to transform these unstructured log messages into a structured format, the classic log parsing technique,

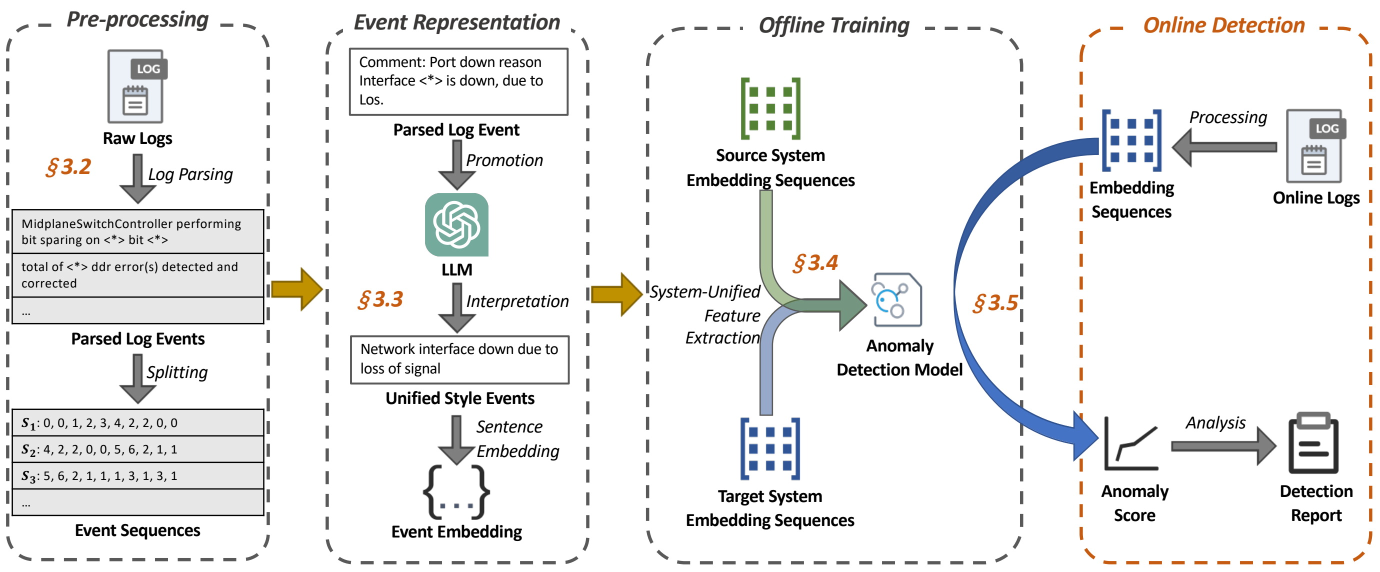

Pre-processing
LOG
Raw Logs
§3.2
Log Parsing
MidplaneSwitchController performing bit sparing on <*> bit <*>
total of <*> ddr error(s) detected and corrected
...
Parsed Log Events
Splitting
S_1: 0, 0, 1, 2, 3, 4, 2, 2, 0, 0
S_2: 4, 2, 2, 0, 0, 5, 6, 2, 1, 1
S_3: 5, 6, 2, 1, 1, 1, 3, 1, 3, 1
...
Event Sequences
Event Representation
Comment: Port down reason Interface <*> is down, due to Los.
Parsed Log Event
Promotion
LLM
§3.3
Interpretation
Network interface down due to loss of signal
Unified Style Events
Sentence Embedding
Event Embedding
Offline Training
Source System Embedding Sequences
§3.4
System-Unified Feature Extraction
Anomaly Detection Model
Target System Embedding Sequences
§3.5
Online Detection
Processing
Embedding Sequences
Online Logs
Analysis
Anomaly Score
Detection Report

Fig. 1: Framework of LogSynergy.

Drain [31], is employed. Drain effectively extracts templates from log messages and maps each log message to its corresponding template. This process converts the unstructured raw logs into structured log events and their associated parameters. Then, LogSynergy splits the continuous log messages into log sequences using a sliding window approach.

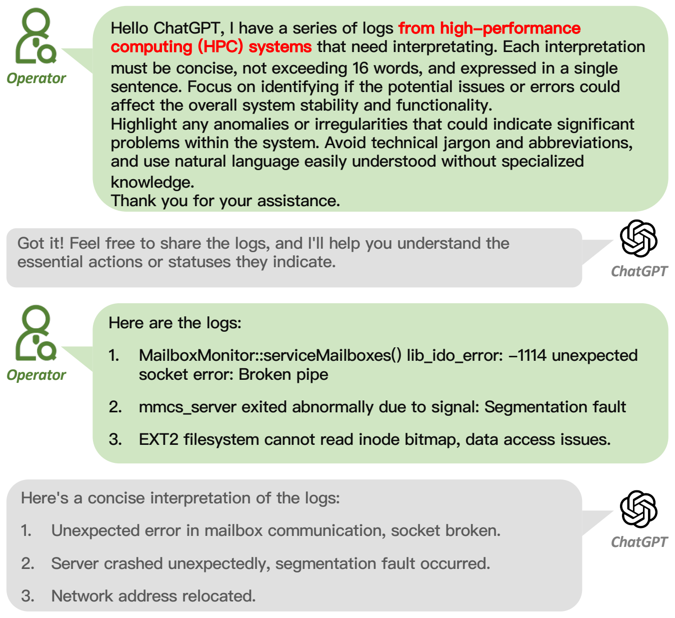
By converting raw logs into structured log events and parameters and segmenting them into log sequences, the data is prepared for subsequent stages such as LLM-based event interpretation, event embedding, and offline training.

C. Event Representation

To obtain a more accurate and unified log representation, LogSynergy first utilizes LEI to interpret log events. Then, it uses a pre-trained model to transform these interpretations into feature vectors within a unified feature space.

LLM-based Event Interpretation (LEI): In recent years, pre-trained LLMs have shown increasing capabilities and have been widely used in log analysis [32]–[35]. These models can analyze log messages and extract precise information across different systems. To reduce the syntax differences, we propose an event interpretation method based on LLMs, specifically using ChatGPT 4o model [36]. This method interprets log messages from various systems uniformly, focusing on the essential information common to them.

By leveraging ChatGPT, LogSynergy can effectively address the differences in log syntax across systems. Operators interact with ChatGPT through prompts, guiding it to interpret log messages semantically. Given the enormous volume of log messages, it is impractical to interpret each one individually. Therefore, LogSynergy uses log parsing methods to extract log events, reducing the number of log messages that ChatGPT needs to interpret. Specifically, LogSynergy selects a representative log message from the messages corresponding to the same log event.



Hello ChatGPT, I have a series of logs from high-performance computing (HPC) systems that need interpreting. Each interpretation must be concise, not exceeding 16 words, and expressed in a single sentence. Focus on identifying if the potential issues or errors could affect the overall system stability and functionality.
Highlight any anomalies or irregularities that could indicate significant problems within the system. Avoid technical jargon and abbreviations, and use natural language easily understood without specialized knowledge.
Thank you for your assistance.

Got it! Feel free to share the logs, and I'll help you understand the essential actions or statuses they indicate.

Here are the logs:
1. MailboxMonitor::serviceMailboxes() lib_ido_error: -1114 unexpected socket error: Broken pipe
2. mmcs_server exited abnormally due to signal: Segmentation fault
3. EXT2 filesystem cannot read inode bitmap, data access issues.

Here's a concise interpretation of the logs:
1. Unexpected error in mailbox communication, socket broken.
2. Server crashed unexpectedly, segmentation fault occurred.
3. Network address relocated.

Fig. 2: An example dialogue process of event interpretation. (System information in red.)

During interactions with ChatGPT, operators construct specific prompts and input typical logs. A general format of the prompts is shown in Fig.2. These prompts typically include a brief description of the system type and keywords about the log event to provide sufficient context for accurate interpretation. Due to the strong capabilities of the ChatGPT 4o model, it can accurately interpret logs from many systems without the need for major adjustments to the prompts. Therefore, in most cases, the prompt only needs to specify the source of the logs. As demonstrated in Fig.2, for logs from HPC systems, we specify in the first sentence that the data comes from an HPC system to enhance interpretation accuracy.

For example, consider an original log message: “Comment:

Port down reason Interface <∗> is down, due to Los." ChatGPT interprets this as "Network interface down due to loss of signal." In this process, ChatGPT understands the abbreviation "Los" and expands it to "loss of signal," clarifying the log message. Moreover, the interpretation uses a more standardized syntax while retaining the necessary information. This enables LogSynergy to effectively understand the log semantics in subsequent steps, reducing the impact of syntax differences across diverse log formats.

However, LEI also has its limitations. The quality of log interpretations heavily depends on the LLM's capabilities and the quality of the prompts. A notable challenge is the hallucination phenomenon commonly observed in pre-trained LLMs [37], where interpretations may contain fabricated or incorrect information. To address this issue, operators need to review the interpretations after generation to identify potential errors. When necessary, interpretations can be regenerated to ensure accuracy and reliability.

Event Embedding: To extract the semantic information from the log interpretations, LogSynergy uses a pre-trained model [38] to embed these log interpretations into a feature space. Specifically, LogSynergy inputs the log interpretations into the pre-trained model and obtains the outputs from the final layer of the model's feature extractor. These outputs (*i.e.,* event embeddings) capture the semantic information of the log events. This embedding technique has been widely adopted in previous works [20], [24]. It is important to note that the selection and use of pre-trained models for event embedding is not our primary contribution. Various pre-trained models, such as BERT [39] and GPT-2 [40], can be utilized.

D. Offline Training

In offline training, LogSynergy uses transfer learning and samples from both source and target systems to train an anomaly detection model. LogSynergy utilizes system-unified feature extraction (SUFE) to filter out system-specific features. Subsequently, domain adaptation is applied to minimize the distribution discrepancy of system-unified features between the source and target systems. In the rest of this subsection, we detail offline training through four aspects: network architecture, SUFE, domain adaptation, and model optimization.

1) Network Architecture: The network of LogSynergy consists of several modules: a feature extractor F, an anomaly classifier C_{anomaly}, a system classifier C_{system}, a mutual information module MI, and a domain adaptation module DA. During the online detection phase, only F and C_{anomaly} are used to detect anomalies, while the other modules are involved only during training to optimize the model's performance.

During training, LogSynergy incorporates SUFE and domain adaptation to improve the model's generalization ability on the target system. As shown in Fig.3, SUFE utilizes MI, C_{system}, and C_{anomaly} to decouple system-specific and system-unified features. C_{system} and C_{anomaly} compute predictions for system labels and anomaly labels using the system-specific and system-unified features, respectively, optimizing the feature extraction process from F through their corresponding loss functions. Meanwhile, MI estimates the mutual information between the two feature types, and SUFE minimizes this mutual information to filter out system-specific features, enhancing generalization. Additionally, DA computes the distinguishability between the system-unified features extracted from the source and target domain samples and minimizes it, further improving the model's ability to generalize to the target system.

F is built from multiple layers of a Transformer Encoder. C_{anomaly} and C_{system} are implemented with fully connected layers. MI consists of several linear and activation layers. DA uses linear layers and a gradient reversal layer (GRL) [41].

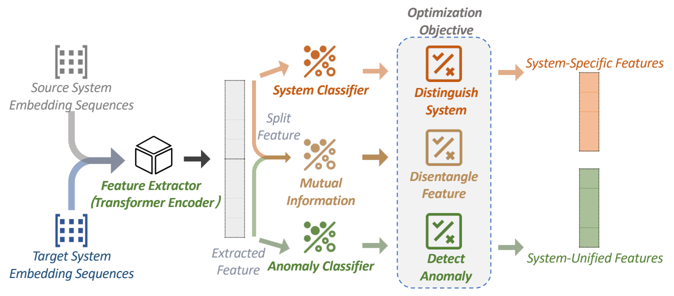


Fig. 3: Multiple optimization objectives of SUFE.

2) System-Unified Feature Extraction (SUFE): SUFE aims to distinguish between system-specific features and system-unified features effectively. Recent advancements in feature disentanglement have shown excellent performance in this area [42]–[44]. Therefore, we designed SUFE based on feature disentanglement.

As shown in Fig.3, SUFE divides the features extracted by the feature extractor F into system-unified features $F_u(x)$ and system-specific features $F_s(x)$. We set that $F_u(x)$ and $F_s(x)$ have the same dimension. Then, to ensure the effectiveness of these features, SUFE uses two classifiers with their respective optimization objectives. The system-specific features are optimized to distinguish the systems using the system classifier C_{system}, which predicts the system label ($\hat{z} = C_{\text{system}}(F_s(x))$). Here, $\hat{z}$ is the predicted probability distribution over K classes, and the true system label z_k is represented as a one-hot encoded vector. The system-unified features are optimized to detect anomalies using the anomaly classifier C_{anomaly}, which predicts the anomaly label ($\hat{y} = C_{\text{anomaly}}(F_u(x))$).

The loss functions for these two objectives are defined as follows:

$$\mathcal{L}_{\text{system}} = -\mathbb{E}_{(x,z)\sim\mathcal{D}}\left[\sum_{k=1}^{K} z_k \log \hat{z}_k\right] \tag{1}$$

$$\mathcal{L}_{\text{anomaly}} = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right] \tag{2}$$

By minimizing $\mathcal{L}_{\text{system}}$ and $\mathcal{L}_{\text{anomaly}}$, respectively, the model can better distinguish between systems using system-specific features $F_s(x)$ and more accurately detect anomalies using system-unified features $F_u(x)$

We introduce a mutual information module MI inspired by MT-MIM [43], [45] to estimate the correlation between system-specific and system-unified features. By minimizing $\mathcal{L}_{\text{MI}}$, SUFE disentangles the system-specific and system-unified features, ensuring that the system-unified features focus solely on anomaly detection, without interference from system-specific features. This disentangling allows the system-unified features to generalize across different systems, thereby improving the model's ability to detect anomalies in the target system.

The mutual information loss is defined as:

$$\mathcal{L}_{\text{MI}} = MI(F_u(x), F_s(x)) \tag{3}$$

To estimate the mutual information, we employ the Contrastive Learning Upper Bound (CLUB) method [46], which provides a tractable and differentiable upper bound for MI. CLUB utilizes several linear and activation layers to compute the difference between the two feature sets, making it a suitable approach for estimating mutual information. For further details on the CLUB method, please refer to the original paper [46].

3) Domain Adaptation: While SUFE effectively disentangles system-specific and system-unified features, the distribution of system-unified features may still differ across systems, which hinders the model's ability to generalize. To address this issue, LogSynergy employs the domain adaptation, which mitigates the distribution discrepancies.

Specificially, LogSynergy employs Dynamic Adversarial Adaptation Network (DAAN) [12], which uses a domain classifier D to measure the distinguishability between the system-unified features extracted from the source and target domains. D computes a cross-entropy loss, which acts as the adversarial objective, pushing the feature extractor to minimize domain-specific signals.

A Gradient Reversal Layer (GRL) is applied during adversarial training to reverse the gradient during backpropagation, enabling the network's modules to be optimized adversarially. This encourages F to extract domain-invariant system-unified features while D learns to distinguish these features across domains. As a result, the model, primarily trained on source domain samples, can generalize well to target domain samples, ensuring effective transfer learning with minimal target domain data.

The domain adaptation loss function is given by:

$$\begin{aligned}\mathcal{L}_{DA} = -\mathbb{E}_{(x,d_{label})\sim\mathcal{D}}\Big[& d_{label}\log D(\mathcal{R}(F_u(x))) \\ &+ (1-d_{label})\log(1-D(\mathcal{R}(F_u(x))))\Big]\end{aligned} \tag{4}$$

Here, d_{label} represents the domain label (0 for the source domain and 1 for the target domain), and $\mathcal{R}(F_u(x))$ denotes the feature extraction with the gradient reversal applied during backpropagation by the GRL. By minimizing $\mathcal{L}_{\text{DA}}$, the model learns to produce system-unified features that are less distinguishable between the source and target domains, thus ensuring better generalization across systems. For a more detailed explanation of DAAN and its application to domain adaptation, please refer to the original paper [12].

4) Model Optimization: During the model optimization process, we combine the loss functions from SUFE and domain adaptation to form the total loss function. Each component of the total loss function plays a critical role in ensuring that the model generalizes well to the target system by balancing different optimization goals. The total loss function is a weighted sum of these loss functions:

$$\mathcal{L} = \mathcal{L}_{\text{system}} + \mathcal{L}_{\text{anomaly}} + \lambda_{\text{MI}}\mathcal{L}_{\text{MI}} + \lambda_{\text{DA}}\mathcal{L}_{\text{DA}} \tag{5}$$

where λ_{MI} represents the weight coefficient for the mutual information loss, and λ_{DA} represents the weight coefficient for the domain adaptation loss.

E. Online Detection

During the online detection phase, LogSynergy uses the model trained offline to detect anomalies in log sequences recorded by the target system. First, LogSynergy processes the logs into event sequences using the same method as offline log processing. Then, it maps the events into event embeddings. When a new log event appears, LogSynergy maps the new log event into an event embedding. The embedding sequences are then input into the model trained offline to obtain anomaly probabilities distributed between 0 and 1. If the anomaly probability exceeds the threshold of 0.5, it is considered an anomaly, and an anomaly report is generated based on event interpretation and anomaly score.

IV. Evaluation

A. Experimental Design

1) Datasets: We conduct experiments using two distinct groups of log datasets. The first group includes three widely used public log datasets: BGL, Spirit, and Thunderbird [5], which we selected following the related works [15], [18], [22], [28]. From these datasets, we use specific subsets. The second group includes three software system log datasets from the production environments of a top-tier global ISP collected in 2023. Detailed information about these datasets is provided in Table III. For the raw log files, we segmented them into log sequences using a sliding window with a step size of 5 and a window length of 10. LogSynergy and the baseline methods used these log sequences as samples during the evaluation process, feeding them into the model.

TABLE III: Detail of the datasets.

Datasets	# of logs	# of log sequences	# of anomalies
BGL	1,356,817	271,362	29,092
Spirit	4,783,733	956,745	8,857
Thunderbird	700,005	140,000	5,946
System A	2,166,422	433,014	886
System B	877,444	175,481	296
System C	691,433	137,258	5,170

For each group of datasets, we set one dataset as the target system and the others as the source systems. From each source system, we select 50,000 sequences for training, and for the target system, we select 5,000 sequences for training. Based on previous research [47], randomly selecting samples from datasets for training can introduce unfair biases due to data leakage. To mitigate this, we adopt a continuous sampling strategy for training samples in the target system. Specifically, we use the former portion of the dataset for training and the latter for testing. This method aligns the experiment more closely with real-world conditions, ensuring that the training process accurately reflects real production environments and reduces the risk of data leakage.

2) Baselines: To fairly compare LogSynergy with existing methods, we choose nine baseline methods listed in Table IV and Table V. These methods have different settings, so they select samples from the training set in various ways. DeepLog [13] and LogAnomaly [14] are unsupervised learning methods that utilize all normal samples from the target system in the training set. PLELog [19] is a semi-supervised method that uses 50% of the normal samples and the remaining unlabeled samples from the training set. SpikeLog [22] is a weakly supervised method that uses 98% of the anomalous samples and the remaining unlabeled samples from the training set. NeuralLog [24] and LogRobust [23] employ supervised learning with all training set samples. PreLog [28] is a pre-trained method for anomaly detection that utilizes the samples from the source systems for pre-training and the samples from the target system for prompt tuning. LogTAD [4] is an unsupervised cross-system anomaly detection method that utilizes all normal samples from the training set. LogTransfer [2], MetaLog [1], and LogSynergy (our method) are supervised cross-system anomaly detection methods that utilize all samples from the training set. Additionally, all the above baselines use the same testing sets as LogSynergy for a fair comparison.

Moreover, all parameters for the baseline methods and LogSynergy were selected based on the optimal F1-Score for fair comparison. For the model configurations, DeepLog and LogAnomaly use two LSTM [48] layers with 128 hidden units and a top-k value of 9. PLELog uses a single LSTM layer with 100 hidden units. SpikeLog uses a single Spiking Neural Network [49] layer with 128 hidden units and an additional LSTM layer with 32 hidden units. NeuralLog uses a single Transformer Encoder [30] layer with an embedding dimension of 768 and a feedforward layer dimension of 2048. LogRobust uses two Bi-LSTM [50] layers with 128 hidden units. MetaLog uses two GRU [51] layers, each with 100 hidden units. Both LogTransfer and LogTAD use two LSTM layers with 128 hidden units.

3) Evaluation Metrics: We select precision, recall, and F1-Score as evaluation metrics, where precision is defined as the ratio of true positive predictions to the total number of positive predictions made, defined by $Precision = \frac{TP}{TP+FP}$; recall is the ratio of true positive predictions to the total actual positives, defined by $Recall = \frac{TP}{TP+FN}$; and F1-Score is the harmonic mean of precision and recall, defined by $F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$. System logs will be defined as anomalous if their anomaly score exceeds the predefined threshold. For all models employing classification, we set the classification threshold at 0.5 to ensure a fair comparison.

4) Implementation Details: We conduct all the experiments on a cloud platform provided by a top-tier global Internet Service Provider (ISP). It provides GPU cloud services with Tesla V100, 16-core CPU, and 32GB DRAM memory. We implement LogSynergy with Python 3.8.15, PyTorch 1.8, transformers 4.35.0, and scikit-learn 1.3.2. Our code is open-sourced at https://anonymous.4open.science/r/LogSynergy-320D/. In our experiments, LogSynergy utilizes a six-layer Transformer Encoder. The number of attention heads is set to 12, and the feed-forward network that processes the output of the multi-head self-attention mechanism has a dimensionality of 2048. We use the AdamW optimizer [52] to train the Transformer Encoder-based model in LogSynergy, with a learning rate of 1×10^{-4}. We set the mini-batch size to 1024 during training and train the models for ten epochs. The hyperparameters are set as follows: $\lambda_{\text{DA}} = 0.01$ and $\lambda_{\text{MI}} = 0.01$. The number of samples used for training is $n_s = 50,000$ from the source system and $n_t = 5,000$ from the target system. Both λ_{MI} and n_t are evaluated in §IV-C. To accelerate model training, we enable fp16 (half-precision) [53] training using the transformers library's built-in support [54]. By using fp16, we reduce memory consumption and increase computational efficiency on GPUs, allowing for faster training times without significantly sacrificing model performance.

B. The Overall Performance

As listed in Table IV and Table V, LogSynergy's F1-Score significantly outperforms all baseline methods. The unsupervised methods, DeepLog [13] and LogAnomaly [14], utilize only a small number of normal samples from the target system for training. Consequently, they cannot learn enough normal patterns, leading to the misclassification of new patterns as anomalies. This results in high recall but low precision, yielding lower F1-scores. PLELog [19] is designed for single-system anomaly detection and cannot bridge the gap between the target and source systems. Moreover, the limited number of target system samples leads to poor adaptation, causing many normal samples to be misclassified as anomalies. This results in low precision despite high recall in multiple experiments.

SpikeLog, NeuralLog, and LogRobust are weakly supervised or supervised single-system anomaly detection methods. Their performance depends on the similarity between the source and target systems as well as their feature engineering, resulting in significant performance variations across different target systems. For instance, in experiments with BGL, System B, and System C as the target systems, both NeuralLog and SpikeLog exhibit poor F1-scores due to the low similarity between the source and target systems. In contrast, LogRobust, with its stronger robustness, performs better than SpikeLog and NeuralLog in these experiments. However, all their performances fall short compared to LogSynergy.

PreLog focuses on pre-training models to optimize semantic representations, rather than directly learning anomaly

TABLE IV: The Precision (P), Recall (R), and F1-Score(F1) on BGL, Spirit, and Thunderbird (One used as the target system and the others as the source).

Method	Type	BGL			Spirit			Thunderbird		
		P(%)	**R(%)**	**F1(%)**	**P(%)**	**R(%)**	**F1(%)**	**P(%)**	**R(%)**	**F1(%)**
DeepLog [13]	Unsupervised	10.77	**100**	19.44	0.99	**100**	1.95	4.60	**100**	8.79
LogAnomaly [14]		11.77	**100**	21.06	10.99	**100**	19.80	25.61	**100**	40.78
PLELog [19]	Semi-supervised	11.16	38.66	17.33	1.17	89.98	2.31	5.14	97.38	9.77
SpikeLog [22]	Weakly-supervised	27.92	51.09	22.10	33.79	31.53	32.62	60.66	68.73	64.44
NeuralLog [24]	Supervised	**100**	2.01	3.95	37.33	73.66	49.55	79.02	99.83	88.21
LogRobust [23]		44.75	46.67	45.69	19.83	25.97	22.49	69.34	97.49	81.04
PreLog [28]	Pre-trained	72.81	68.63	70.66	0	0	0	79.51	96.87	87.34
LogTAD [4]	Unsupervised Cross-System	10.27	78.59	18.16	1.33	85.55	2.62	6.33	99.30	11.9
LogTransfer [2]	Supervised Cross-System	13.41	2.70	4.50	26.39	18.85	21.99	85.14	71.69	77.84
MetaLog [1]		15.23	91.12	26.10	3.13	15.51	4.50	3.00	3.39	3.18
LogSynergy		97.43	72.83	**83.35**	**88.91**	92.41	**90.62**	**96.23**	99.83	**97.99**

TABLE V: The Precision (P), Recall (R), and F1-Score(F1) on System A, System B, and System C (One used as the target system and the others as the source).

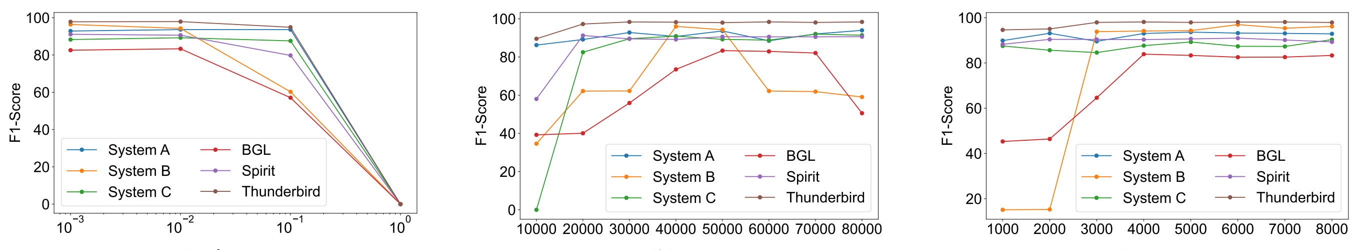
Method	Type	System A			System B			System C		
		P(%)	**R(%)**	**F1(%)**	**P(%)**	**R(%)**	**F1(%)**	**P(%)**	**R(%)**	**F1(%)**
DeepLog [13]	Unsupervised	0.64	**100**	1.28	0.16	**100**	0.32	4.02	**100**	7.73
LogAnomaly [14]		1.04	99.89	2.06	0.86	**100**	1.71	5.13	98.99	9.75
PLELog [19]	Semi-supervised	3.24	89.29	6.26	0.18	91.64	0.35	6.68	89.93	12.42
SpikeLog [22]	Weakly-supervised	31.81	93.11	47.43	0.26	18.75	0.51	2.13	87.38	4.16
NeuralLog [24]	Supervised	29.57	80.40	43.24	**100**	13.74	24.16	**100**	0.82	1.63
LogRobust [23]		32.04	91.39	47.45	93.40	37.64	53.66	88.97	85.98	87.45
PreLog [28]	Pre-trained	0	0	0	0.07	84.82	0.14	0	0	0
LogTAD [4]	Unsupervised Cross-System	5.28	96.41	10.01	11.94	99.62	21.33	35.70	99.06	52.48
LogTransfer [2]	Supervised Cross-System	31.72	91.04	47.05	24.00	13.69	17.43	0	0	0
MetaLog [1]		3.22	99.77	6.23	13.79	36.12	19.96	78.31	80.31	79.30
LogSynergy		**92.31**	94.99	**93.63**	91.73	96.96	**94.27**	92.22	86.49	**89.26**

detection. While it performs well for systems like BGL and Thunderbird, where new anomalies share semantic similarities with existing ones, its performance drops significantly for other systems, with F1-scores nearing zero. This highlights the need for sufficient target system samples during PreLog's fine-tuning process.

Similar to DeepLog and LogAnomaly, LogTAD learns normal log patterns to detect anomalies. However, there are limitations when applying LogTAD to the target system. On the one hand, due to the limited number of samples from the target system, LogTAD cannot learn enough normal patterns specific to the target system. On the other hand, although LogTAD employs domain adaptation to bridge the differences between the source and target systems, it cannot reduce the syntax differences. These syntax differences typically influence its feature engineering. Consequently, the normal patterns from the source systems cannot be effectively applied to the target system. This results in a high number of false positives, as many new patterns are incorrectly identified as anomalies, leading to low precision and F1-scores.

Among the supervised cross-system log anomaly detection methods, LogTransfer achieves a high F1-Score on Thunderbird, and MetaLog achieves a high F1-Score on System C. However, their performance is notably lower on other systems. LogTransfer is suited for similar systems and struggles with dissimilar systems. For example, in BGL and System C, LogTransfer gets a considerably lower F1-Score than other systems because BGL and System C are significantly different from their source systems in syntax and patterns. MetaLog is designed for target systems with few anomalies. However, it requires a sufficient number of normal samples from the target system. Due to the limited number of samples from the target system, MetaLog achieves a low F1-Score, with less than 10% in three sets of experiments. However, MetaLog achieved an F1-Score of 79.3% in System C, indicating that its performance is unstable. MetaLog can achieve a high F1-Score when certain anomaly patterns are well modeled by its feature engineering.

LogSynergy reduces the log syntax differences and filters out the system-specific knowledge, enabling the effective transferring knowledge from the source systems to the target system. These results demonstrate that LogSynergy can train



(a) λ_{MI} (b) n_s (c) n_t

Fig. 4: F1-Score with different hyper-parameter values: (a) λ_{MI}, (b) n_s and (c) n_t across different datasets.

a log anomaly detection model for a new system rapidly with high performance.

C. *Influence of Hyper-Parameters*

In this section, we evaluate the impact of hyperparameters on LogSynergy, including the weight coefficient of $\mathcal{L}_{\text{MI}}$ (i.e., λ_{MI}), as well as the number of source system samples and labels (n_s) and target system samples and labels (n_t) during training. In the evaluations above, we set λ_{MI}, n_s, and n_t to their respective default values.

For λ_{MI}, we assess different values, including 0.001, 0.01, 0.05, 0.1, and 0.5. As shown in Fig. 4a, we find that the performance remains stable when λ_{MI} is small but decreases significantly as λ_{MI} increases. As mentioned in §III-D2, when λ_{MI} is too large, the model prioritizes reducing the correlation between system-specific and system-unified features, neglecting the differences in their classification characteristics, which leads to suboptimal feature disentanglement. Conversely, better performance is generally achieved when λ_{MI} is small. Specifically, setting λ_{MI} to 0.01 results in the highest F1-Score on four target systems and near-highest scores on two others. Therefore, the default value of λ_{MI} is set to 0.01.

For n_s, we set eight values ranging from 10,000 to 80,000, with steps of 10,000. Generally, we find that performance improves with more samples and labels, stabilizing around 50,000. However, in some cases where poor-quality samples were randomly selected, increasing the sample size did not necessarily improve anomaly detection, suggesting that the benefit of more samples depends on their quality.

For n_t, we set eight values ranging from 1,000 to 8,000, with steps of 1,000. Generally, the performance of the model improves as more target system samples and labels are included in the training process. As shown in Fig. 4c, the F1-Score increases significantly as n_t grows initially and begins to stabilize around 4,000. This indicates that LogSynergy can achieve acceptable and stable performance with only a few target system samples and labels, making it well-suited for log anomaly detection in new systems.

D. *Ablation Study*

We conduct ablation experiments on the six datasets mentioned above to evaluate the effectiveness of transfer learning and two essential modules in LogSynergy: LLM-based event interpretation (LEI) and system unified feature extraction (SUFE).

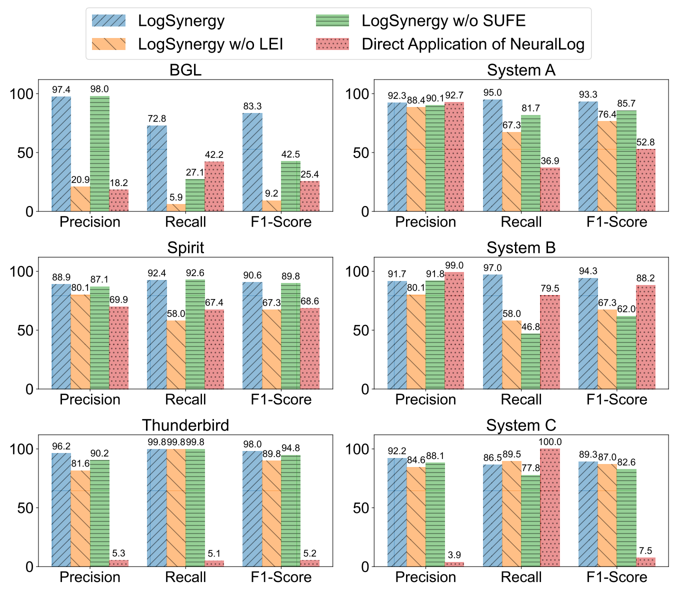


Fig. 5: Ablation study results of LEI, SUFE and transfer learning.

1) Effectiveness of LEI: Our proposed LogSynergy uses LEI to interpret log events, thereby unifying the log syntax from different systems. To evaluate the effectiveness of this method, we remove LEI from LogSynergy and map log events directly to the feature space instead of using interpreted logs. We denote LogSynergy with LEI removed as “LogSynergy w/o LEI”, and its performance is shown in Fig.5.

The results presented in Fig.5 demonstrate that LEI greatly improves the performance of LogSynergy, especially on the BGL, where it improves the F1-Score by up to 74%. This method enhances the performance of transfer learning by making logs with similar content from different systems more easily transferable through unified log syntax. These findings indicate that our method of unifying different log syntax through LEI can effectively reduce the differences between logs from various systems and substantially enhance transfer learning performance.

2) Effectiveness of SUFE: In our proposed method, we use SUFE to filter out system-specific features and retain system-unified features, so that the extracted features are more applicable to the target system. To evaluate the effectiveness of SUFE, we remove SUFE from LogSynergy and train the log anomaly detection model using only domain adaptation. We denote LogSynergy with SUFE removed as “LogSynergy w/o SUFE”, and its performance is shown in Fig.5.

Fig.5 illustrates that SUFE positively impacts the performance of LogSynergy. Across experiments on six datasets, SUFE consistently improved the F1-Score on each dataset. By extracting system-shared features, SUFE effectively reduces the influence of system-specific features on feature distribution, thereby enhancing the model’s generalization performance on the target system. This improvement in generalization enables the model trained by LogSynergy to perform better in anomaly detection on the target system’s logs.

3) Effectiveness of Transfer Learning: LogSynergy applies transfer learning to transfer log anomaly detection knowledge from the source systems to the target system, addressing the lack of data in the target system. To evaluate the effectiveness of transfer learning, we select the state-of-the-art single-system log anomaly detection method, NeuralLog, and train the model using only the data from the source system, directly evaluating it on the target system’s logs. We refer to this method, which does not use transfer learning, as the “direct application of NeuralLog”, and its performance is shown in Fig.5.

Fig.5 illustrates that transfer learning significantly impacts log anomaly detection performance on the target system. In experiments across six datasets, the performance of the direct application of NeuralLog was consistently lower than that of LogSynergy. The F1-Score of direct application of NeuralLog reached 88.2% for System B, while the performance on other datasets was significantly lower than LogSynergy. This experiment demonstrates that even though the source and target systems have similarities, models trained directly on the source system typically do not perform well on the target system, highlighting the necessity of transfer learning.

E. Threats to Validity

1) External Threats: In evaluating LogSynergy, we selected portions of logs from three public datasets and real logs from a top-tier global ISP. Based on the analysis of these real logs, we found that anomalies occur at a very low frequency in the real world. For instance, some logs with negative semantics, such as frequent login failures, are not considered anomalies in practice. LogSynergy is designed for real-world scenarios, learning anomaly classification from source systems. However, it can also be affected by low-quality or misclassified log anomalies. When the quality of the logs is poor, and logs that frequently appear but do not significantly impact the software system are treated as anomalies, LogSynergy may learn from these misclassified anomalies, leading to a degradation in performance.

2) Internal Threats: As mentioned in §III-C, due to the hallucination problem of LLMs, there might be occasional biased or incorrect log interpretations. These inaccuracies can influence the effectiveness of subsequent log representations and the final performance of the anomaly detection model. For example, if an LLM misinterprets a critical log event, the anomaly detection could be flawed. However, given that the number of distinct log events is limited (usually only a few hundred), manually reviewing the LLM-generated log interpretations for these events is feasible and requires less human effort. This manual check can mitigate the impact of potential hallucinations, ensuring the reliability of the anomaly detection process.

V. Lesson Learned

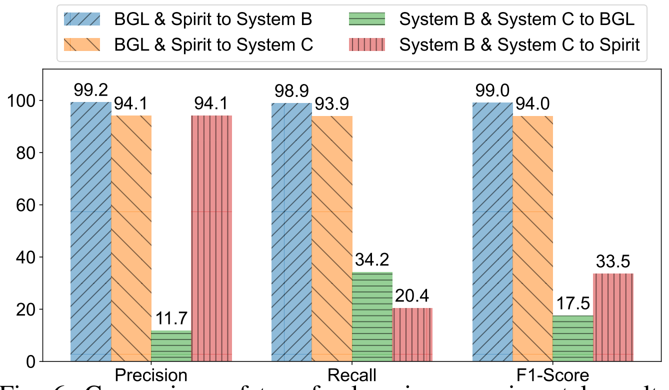


Fig. 6: Comparison of transfer learning experimental results across different source and target systems.

In this section, we discuss and analyze some shortcomings of LogSynergy. LogSynergy is based on the assumption that there should be a certain degree of similarity between the source and target systems. Although LogSynergy achieves uniformity in log syntax and features through two key components, LEI and SUFE, software systems often have significant differences that can impact transfer learning performance. Specifically, when two systems differ significantly in functionality, transfer learning performance may degrade. Similarly, if the source system includes some functionalities or anomalies of the target system, LogSynergy will exhibit excellent anomaly detection performance. Conversely, if the anomalies in the source system do not cover those in the target system or if there are significant differences between the target and source systems, LogSynergy cannot guarantee the effectiveness of transfer learning.

To verify these two scenarios, we select logs from two systems in each group for mutual transfer learning. Specifically, we use BGL and Spirit from the first group as source systems and transferred them to System B and System C from the second group respectively; similarly, we use System B and System C from the second group as source systems and transferred them to BGL and Spirit from the first group. The performance comparison of these four sets of experiments is shown Fig.6. We find that transferring from BGL and Spirit to System B and System C results in high performance, with F1-scores of 99% and 94%, respectively. Conversely, transferring from System B and System C to BGL and Spirit results in lower performance.

We think the reason is that, as supercomputer systems, BGL and Spirit’s logs typically contain logs from various components and multiple anomaly types, possessing rich knowledge for anomaly detection. Therefore, when transferring to the relatively simpler System B and System C, they can cover most of the anomalies in these systems. On the other hand, the performance is unsatisfactory since System B and System

C cannot cover the anomalies and functionalities of BGL and Spirit. Specifically, when Spirit is the target system, its precision is high, but recall is low, indicating that some anomalies have yet to occur in System B and System C. For BGL, both precision and recall are low, reflecting a significant shortage in anomaly detection performance.

VI. Deployment and Application

A. Workflow of Deployment

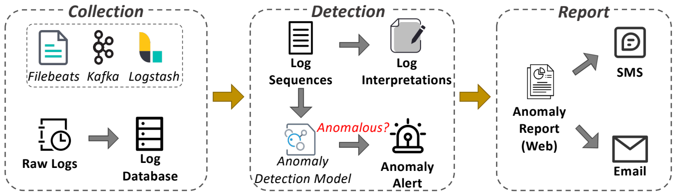


Fig. 7: The workflow of LogSynergy in the production environment.

To investigate the effectiveness of LogSynergy in real-world applications, we deploy LogSynergy on the cloud platform provided by the top-tier global ISP mentioned in §IV. LogSynergy is deployed as an application within a Docker container equipped with a V100 GPU.

As shown in Fig.7, the workflow of LogSynergy is divided into three stages:

Collection: Logs are collected by Filebeat [55] in real-time, transferred to Kafka [56] for buffering, and parsed by LogStash [57]. Filebeat is deployed on distributed systems to gather raw logs, which are then formatted into a unified structure by LogStash. The logs are split into sequences using a sliding window approach, where each window consists of 10 logs with a 5-step shift. This process standardizes logs across systems, enabling efficient anomaly detection in later stages.

Detection: Due to the large volume of logs, we use a pattern-matching method for online detection. When a new log sequence is generated, it is first matched against a pattern library of historical anomalies from LogSynergy. If a new pattern is detected, the sequence is processed by the offline-trained LogSynergy model for anomaly detection, minimizing computational overhead from redundant log patterns.

Report: Upon detecting an anomaly, LogSynergy generates a detailed report combining the log interpretation from LEI and the anomaly alert. The report includes the original log sequence, interpretations, and relevant metadata (timestamps, system identifiers). Reports are sent to operations engineers via SMS and email for timely alerts, ensuring prompt response and resolution.

B. Details of Deployment

This subsection provides details about key aspects of LogSynergy's operation, including labeling procedures, the use of LLMs for generating and reviewing interpretations, and the time required for model training.

1) Labeling Process: The labels in the new systems are manually annotated by operators. To ensure the reliability of the annotations, two operators independently label each sequence. In cases of disagreement, a third operator is consulted to resolve the conflict. This collaborative approach minimizes labeling errors and ensures high-quality training data for the new system.

2) LLM-Generated Interpretations and Error Review: LogSynergy utilizes the ChatGPT 4o model to generate interpretations for log templates, with API interactions ensuring that this process is completed rapidly. In our experiments, generating interpretations for a new dataset typically takes less than a minute, as only a few hundred log templates require processing.

As mentioned in §III-C, all LLM-generated interpretations must be reviewed to ensure accuracy, with the focus of the review being on detecting errors in format and length rather than semantic correctness. The interpretations can be regenerated when format errors are found to ensure compliance with the required format. Since a typical dataset contains only a few hundred templates, and each template is reviewed once for errors, the review process is efficient. Operators can complete this task within ten minutes, ensuring the overall workload remains manageable.

3) Training Time: The training process for LogSynergy is highly efficient, taking approximately 10 minutes. This speed is achieved by using only 50,000 sequences from each mature system and 5,000 sequences from the new system. Additionally, mixed-precision training is employed to further accelerate the process. This efficient training pipeline ensures that LogSynergy can be rapidly deployed in real-world scenarios.

C. Effectiveness of Depolyment

LogSynergy has been deployed for over three months in the CDMS of a top-tier global ISP, during which more than 100 new software systems were deployed or updated. Each system requires a corresponding log-based anomaly detection model, and given the frequency of updates, applying supervised methods across all systems becomes impractical. As a result, unsupervised and rule-based methods are primarily used, but they suffer from a high reliance on operator-defined rules, leading to false negatives. The process of summarizing rules is time-consuming and costly, making it difficult to quickly generate enough rules to cover anomalies in new systems.

In contrast, LogSynergy offers significant advantages in both deployment efficiency and performance.

1) Deployment Efficiency: LogSynergy enables rapid deployment of anomaly detection models for new software systems. Traditional rule-based methods require engineers to generate over ten anomaly detection rules for each system, a process that takes 1-2 weeks per rule. This rule accumulation involves analyzing issues, identifying patterns, and validating rules, which leads to delays and substantial manual effort.

LogSynergy, on the other hand, requires only tens of thousands of logs, and typically completes model training and initial deployment within a few days. The log collection can be done in a day, and manual labeling typically takes just a few hours. This drastically reduces the manual effort and delays, enabling faster deployment of log-based anomaly detection models. According to our statistics, the deployment time using LogSynergy is reduced by over 90% compared to traditional rule-based methods.

2) Performance: LogSynergy significantly improves the performance by addressing the limitations of rule-based methods. While rule-based approaches offer high precision, they can only detect predefined anomalies, resulting in low recall. This is especially problematic for new systems with limited data, where it takes considerable time to accumulate enough rules for adequate recall.

In contrast, LogSynergy leverages transfer learning from mature systems to adapt to new, unknown anomalies. This allows it to detect a wider range of anomalies while maintaining high precision, thereby significantly improving recall and overall performance. Although production data are rarely fully labeled, we could not precisely quantify the performance improvement. However, based on our analysis of false positives and false negatives, we conclude that LogSynergy outperforms the rule-based methods.

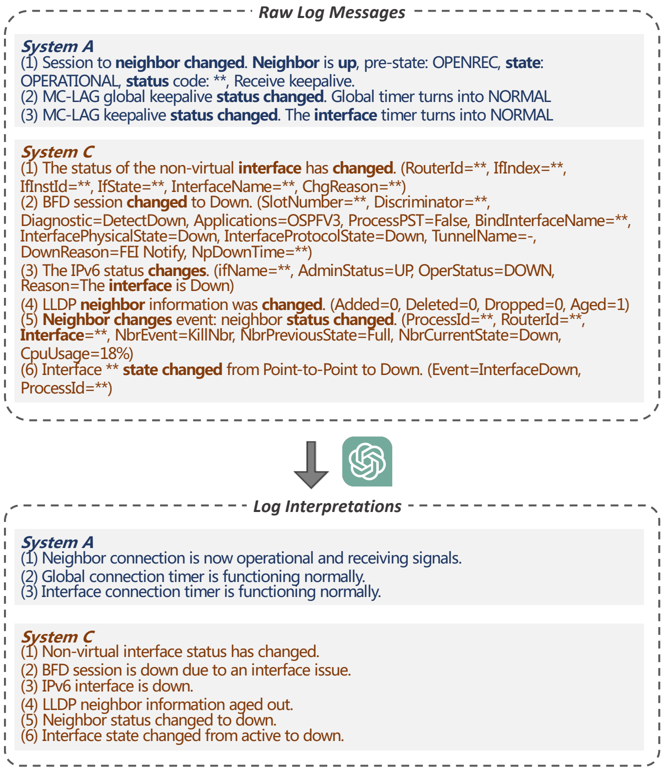
D. Case Study

The quantitative evaluation of the deployed LogSynergy has been discussed in §IV. In this section, we perform a case study to further demonstrate the advantage of LogSynergy. In §IV, we find that LogTransfer performs the best among cross-system log anomaly detection methods in System A. Therefore, we select a false positive case from LogTransfer for detailed analysis. System A is the new system, while System C is the mature system. The upper part of Fig.8 depicts the raw log messages from System A, including highlighted words relevant to the false positive case. To highlight the issue, we remove some repetitive or non-essential log messages from the sequence.

Since this pattern does not appear in the training samples of System A, it relies on the training samples from the mature system used for transfer learning. By analyzing these training samples, we identify the closest match in System C. In System C, this sample is anomalous, indicating multiple interfaces and sessions experiencing consecutive anomalous state changes. However, in System A, the sample is normal, representing typical state changes.

The issue arises because the log messages from System A share a high degree of similarity with those from System C. This similarity is especially pronounced in the words used within the log messages. We highlight these similar words in the raw log messages shown in Fig.8. LogTransfer, utilizing Word2Vec [58] and GloVe [59] to generate log representations, interprets these similar words as similar features. Consequently, the model mistakenly identifies the normal sample from System A as being similar to the anomalous sample from System C. The misinterpretation of the log representations leads to the false positive.

LogSynergy, however, uses the LEI method to obtain log interpretations, effectively converting log messages into more accurate log representations. As shown in the lower part of Fig.8, the log interpretations focus on the essential common information across logs with different syntax (such as current states), ignoring less important details (such as the specifics of changes). As a result, the similarity between the log interpretations of these messages is significantly lower, thus preventing false positives due to misleading similarity.



System A
(1) Session to **neighbor changed**. **Neighbor** is **up**, pre-state: OPENREC, **state**: OPERATIONAL, **status** code: **, Receive keepalive.
(2) MC-LAG global keepalive **status changed**. Global timer turns into NORMAL
(3) MC-LAG keepalive **status changed**. The **interface** timer turns into NORMAL

System C
(1) The status of the non-virtual **interface** has **changed**. (RouterId=**, IfIndex=**, IfInstId=**, IfState=**, InterfaceName=**, ChgReason=**)
(2) BFD session **changed** to Down. (SlotNumber=**, Discriminator=**, Diagnostic=DetectDown, Applications=OSPFV3, ProcessPST=False, BindInterfaceName=**, InterfacePhysicalState=Down, InterfaceProtocolState=Down, TunnelName=-, DownReason=FEI Notify, NpDownTime=**)
(3) The IPv6 status **changes**. (ifName=**, AdminStatus=UP, OperStatus=DOWN, Reason=The **interface** is Down)
(4) LLDP **neighbor** information was **changed**. (Added=0, Deleted=0, Dropped=0, Aged=1)
(5) **Neighbor changes** event: neighbor **status changed**. (ProcessId=**, RouterId=**, **Interface**=**, NbrEvent=KillNbr, NbrPreviousState=Full, NbrCurrentState=Down, CpuUsage=18%)
(6) Interface ** **state changed** from Point-to-Point to Down. (Event=InterfaceDown, ProcessId=**)

System A
(1) Neighbor connection is now operational and receiving signals.
(2) Global connection timer is functioning normally.
(3) Interface connection timer is functioning normally.

System C
(1) Non-virtual interface status has changed.
(2) BFD session is down due to an interface issue.
(3) IPv6 interface is down.
(4) LLDP neighbor information aged out.
(5) Neighbor status changed to down.
(6) Interface state changed from active to down.

Fig. 8: A false positive case in System A caused by a misleadingly "similar" case in System C.

VII. Conclusion

In this paper, we proposed LogSynergy, a novel transfer learning-based framework for log anomaly detection designed to bridge the gap between logs from different software systems. LogSynergy tackles syntax differences and system-specific knowledge through two key innovations: LLM-based event interpretation (LEI) and system-unified feature extraction (SUFE). These innovations improve knowledge transfer and enhance the generalizability of anomaly detection models. LogSynergy has been deployed in the production environment of a top-tier global ISP, where it was evaluated on three real-world datasets and three public datasets. It achieved F1-scores of over 89% on real-world datasets and 83% on public datasets, using only 5000 labeled log sequences from a new system. These results demonstrate the effectiveness of LogSynergy in rapidly deploying anomaly detection models. In the future, we plan to further validate LogSynergy's performance in more diverse scenarios and explore potential enhancements to handle increasingly complex log data.

References

[1] C. Zhang, T. Jia, G. Shen, P. Zhu, and Y. Li, "Metalog: Generalizable cross-system anomaly detection from logs with meta-learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639205

[2] R. Chen, S. Zhang, D. Li, Y. Zhang, F. Guo, W. Meng, D. Pei, Y. Zhang, X. Chen, and Y. Liu, "Logtransfer: Cross-system log anomaly detection for software systems with transfer learning," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 37–47.

[3] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "Transfer log-based anomaly detection with pseudo labels," in *2020 16th International Conference on Network and Service Management (CNSM)*, 2020, pp. 1–5.

[4] X. Han and S. Yuan, "Unsupervised cross-system log anomaly detection via domain adaptation," in *Proceedings of the 30th ACM international conference on information & knowledge management*, 2021, pp. 3068–3072.

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

[6] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, 2024.

[7] L. Batina, B. Gierlichs, E. Prouff, M. Rivain, F.-X. Standaert, and N. Veyrat-Charvillon, "Mutual information analysis: a comprehensive study," *Journal of Cryptology*, vol. 24, no. 2, pp. 269–291, 2011.

[8] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021.

[9] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[10] A. Farahani, S. Voghoei, K. Rasheed, and H. R. Arabnia, "A brief review of domain adaptation," *Advances in data science and information engineering: proceedings from ICDATA 2020 and IKE 2020*, pp. 877–894, 2021.

[11] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. Smola, "A kernel method for the two-sample-problem," *Advances in neural information processing systems*, vol. 19, 2006.

[12] C. Yu, J. Wang, Y. Chen, and M. Huang, "Transfer learning with dynamic adversarial adaptation network," in *2019 IEEE international conference on data mining (ICDM)*. IEEE, 2019, pp. 778–786.

[13] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[14] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, vol. 19, no. 7, 2019, pp. 4739–4745.

[15] L. Ma, L. Cao, P. M. VanNostrand, D. M. Hofmann, Y. Su, and E. A. Rundensteiner, "Pluto: Sample selection for robust anomaly detection on polluted log data," *Proceedings of the ACM on Management of Data*, vol. 2, no. 4, pp. 1–25, 2024.

[16] T. Jia, Y. Wu, C. Hou, and Y. Li, "Logflash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems," in *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021, pp. 80–90.

[17] X. Song, Y. Zhu, J. Wu, B. Liu, and H. Wei, "Adops: An anomaly detection pipeline in structured logs," *Proc. VLDB Endow.*, vol. 16, no. 12, p. 4050–4053, Aug. 2023. [Online]. Available: https://doi.org/10.14778/3611540.3611618

[18] P. Jia, S. Cai, B. C. Ooi, P. Wang, and Y. Xiong, "Robust and transferable log-based anomaly detection," *Proc. ACM Manag. Data*, vol. 1, no. 1, May 2023. [Online]. Available: https://doi.org/10.1145/3588918

[19] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1448–1460.

[20] Z. Yin, X. Kong, and C. Yin, "Semi-supervised log anomaly detection based on bidirectional temporal convolution network," *Computers & Security*, vol. 140, p. 103808, 2024.

[21] C. Duan, T. Jia, H. Cai, Y. Li, and G. Huang, "Afalog: A general augmentation framework for log-based anomaly detection with active learning," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 46–56.

[22] J. Qi, Z. Luan, S. Huang, C. Fung, H. Yang, and D. Qian, "Spikelog: Log-based anomaly detection via potential-assisted spiking neuron network," *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[23] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 807–817.

[24] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 492–504.

[25] S. Hashemi and M. Mäntylä, "Onelog: towards end-to-end software log anomaly detection," *Automated Software Engineering*, vol. 31, no. 2, p. 37, 2024.

[26] A. Vervaet, "Monilog: An automated log-based anomaly detection system for cloud computing infrastructures," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2021, pp. 2739–2743.

[27] J. Qi, S. Huang, Z. Luan, S. Yang, C. Fung, H. Yang, D. Qian, J. Shang, Z. Xiao, and Z. Wu, "Loggpt: Exploring chatgpt for log-based anomaly detection," in *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. IEEE, 2023, pp. 273–280.

[28] V.-H. Le and H. Zhang, "Prelog: A pre-trained model for log analytics," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–28, 2024.

[29] J. Su, C. Jiang, X. Jin, Y. Qiao, T. Xiao, H. Ma, R. Wei, Z. Jing, J. Xu, and J. Lin, "Large language models for forecasting and anomaly detection: A systematic literature review," *arXiv preprint arXiv:2402.10350*, 2024.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[31] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.

[32] Z. Ma, A. R. Chen, D. J. Kim, T.-H. Chen, and S. Wang, "Llmparser: An exploratory study on using large language models for log parsing," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639150

[33] Y. Liu, S. Tao, W. Meng, J. Wang, W. Ma, Y. Zhao, Y. Chen, H. Yang, Y. Jiang, and X. Chen, "Logprompt: Prompt engineering towards zero-shot and interpretable log analysis," *arXiv preprint arXiv:2308.07610*, 2023.

[34] J. Xu, R. Yang, Y. Huo, C. Zhang, and P. He, "Divlog: Log parsing with prompt enhanced in-context learning," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.

[35] J. Xu, Z. Cui, Y. Zhao, X. Zhang, S. He, P. He, L. Li, Y. Kang, Q. Lin, Y. Dang *et al.*, "Unilog: Automatic logging via llm and in-context learning," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–12.

[36] OpenAI, "Chatgpt," 2023, https://www.openai.com/chatgpt.

[37] V. Rawte, A. Sheth, and A. Das, "A survey of hallucination in large foundation models," *arXiv preprint arXiv:2309.05922*, 2023.

[38] HF Canonical Model Maintainers, "distilbert-base-uncased-finetuned-sst-2-english (revision bfdd146)," 2022. [Online]. Available: https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english

[39] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[40] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[41] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *International conference on machine learning*. PMLR, 2015, pp. 1180–1189.

[42] D. Mahapatra, S. Korevaar, B. Bozorgtabar, and R. Tennakoon, "Unsupervised domain adaptation using feature disentanglement and gcns for medical image classification," in *European Conference on Computer Vision*. Springer, 2022, pp. 735–748.

[43] X. Hou, Y. Li, and S. Wang, "Disentangled representation for age-invariant face recognition: A mutual information minimization perspective," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 3692–3701.

[44] X. Jie, X. Zhou, C. Su, Z. Zhou, Y. Yuan, J. Bu, and H. Wang, "Disentangled anomaly detection for multivariate time series," in *Companion Proceedings of the ACM on Web Conference 2024*, 2024, pp. 931–934.

[45] C. Hu, J. Wu, C. Sun, X. Chen, and R. Yan, "Mutual information-based feature disentangled network for anomaly detection under variable working conditions," *Mechanical Systems and Signal Processing*, vol. 204, p. 110804, 2023.

[46] P. Cheng, W. Hao, S. Dai, J. Liu, Z. Gan, and L. Carin, "Club: A contrastive log-ratio upper bound of mutual information," in *International conference on machine learning*. PMLR, 2020, pp. 1779–1788.

[47] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: how far are we?" in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1356–1367. [Online]. Available: https://doi.org/10.1145/3510003.3510155

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[49] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural networks*, vol. 111, pp. 47–63, 2019.

[50] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[51] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 2017, pp. 1597–1600.

[52] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

[53] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham, "Harnessing gpu tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 603–613.

[54] H. Face, "Transformers: State-of-the-art natural language processing for pytorch, tensorflow, and jax," 2020, https://github.com/huggingface/transformers.

[55] B. Elasticsearch, "Filebeat-lightweight shipper for logs (2020)," *URL https://www. elastic. co/products/beats/filebeat. Accessed*, pp. 02–12, 2021.

[56] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, 2011, pp. 1–7.

[57] E. Stack, "Elasticsearch, logstash, kibana— elastic," *URL: https://www.elastic.co/what-is/elk-stack*, 2021.

[58] K. W. Church, "Word2vec," *Natural Language Engineering*, vol. 23, no. 1, pp. 155–162, 2017.

[59] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.