Large Language Model Based on Semantic Compression for Log Anomaly Detection

Xiangyu Zhu, Wenhua Cui, Yuhao Chen, Ye Tao, and Xilong Wang

Abstract—**Efficient text processing is increasingly crucial as data volumes grow exponentially across industries. This paper presents a novel text compression framework, LLM-Distil, that efficiently reduces computational demands while preserving crucial semantic information. The method utilizes template-based preprocessing and knowledge distillation to compress lengthy text sequences, maintaining essential details. Experiments on three benchmark log datasets (HDFS, BGL, and Thunderbird) show that our approach achieves comparable anomaly detection accuracy to existing models, while significantly improving computational efficiency and reducing latency. On average, the framework reduces text length by over 90%, resulting in faster inference times and reduced resource consumption, making it ideal for large-scale applications. Furthermore, our analysis indicates that the Information Retention Rate (IRR) remains above 85% after compression, ensuring the preservation of critical data features. Future work will explore extending this approach to other domains, assessing the impact of data characteristics on compression effectiveness, and optimizing the model for a broader range of anomaly detection tasks. Overall, our results indicate that LLM-Distil has the potential to transform how large language models manage extensive textual inputs in resource-limited environments.**

Index Terms—**LLMs, log sequences, anomaly detection, text compression**

Manuscript received Jan. 24, 2025; revised July. 1, 2025. This work was supported by the National Natural Science Foundation of China (62272093), the Economic and Social Development Research Topics of Liaoning Province (2025-10146-244), and the Postgraduate Education and Teaching Reform Research Project of Liaoning Province (LNYJG2024092).

Xiangyu Zhu is a Ph.D. candidate of School of Electronic and Information Engineering, University of Science and Technology Liaoning, Anshan, China. (e-mail: zhuxiangyu0213@163.com).

Wenhua Cui is a Professor of School of Computer Science and Software Engineering, University of Science and Technology Liaoning, Anshan, China. (Corresponding author to provide phone: +86-133-0422-4928; e-mail: taibeijack@126.com).

Yuhao Chen is a Ph.D. candidate of School of Electronic and Information Engineering, University of Science and Technology Liaoning, Anshan, China. (e-mail: chenyuhao901113@163.com).

Ye Tao is an Associate Professor of School of Computer Science and Software Engineering, University of Science and Technology Liaoning, Anshan, China. (e-mail: taibeijack@163.com).

Xilong Wang is a Postgraduate of School of Electronic and Information Engineering, University of Science and Technology Liaoning, Anshan, China. (e-mail: wang_xl2024@163.com).

I. Introduction

The dynamic evolution of the digital economy is propelled by the advancement of 5G technology alongside the emergence of cutting-edge technologies such as cloud computing, Blockchain, and artificial intelligence. As the proliferation of devices and applications increases, the volume of data, including events and alerts, essential for the operation and maintenance of IT systems, undergoes exponential growth [1]. The manual analysis of these complex data becomes increasingly challenging. System logs, as the predominant form of data in O&M operations, encompass records generated by contemporary network equipment, systems, and service programs during their operational cycles, documenting system status and event information. Log analysis enables the identification of aberrant log sequences, which are crucial for troubleshooting and problem diagnosis. Consequently, log files play a pivotal role in network monitoring, system stability maintenance, performance issue debugging, and software security safeguarding.

Recently, deep learning models, particularly recurrent neural networks (RNNs), have been widely utilized to detect log anomalies due to their ability to model sequential data [2,3,4]. Log data is generated chronologically, with a time dependency often existing between log records. RNNs can capture sequential and time-dependent information to understand the dynamics of log sequences. By training on numerous normal log sequences, RNNs can identify potential anomalies, such as irregular log entry frequency or changes in content and structure. Additionally, RNNs can classify log sequences, distinguishing between normal and abnormal entries. The trained RNN model predicts the next log entry at each time step and assesses its abnormality. However, RNNs have limitations in log anomaly detection. Firstly, their complex time dependencies and gradient propagation can lead to longer training times. Secondly, RNNs may encounter gradient vanishing or explosion problems when processing very long sequences, affecting training effectiveness. Third, while RNNs effectively capture local dependencies, they struggle with global long-distance dependencies. Large language models possess strong sequential modeling capabilities, handle long-distance dependencies effectively, and understand complex log patterns through pretraining and fine-tuning. These models detect subtle anomalies and provide more accurate predictions and classifications. Leveraging the power of large language models improves the accuracy and efficiency of log anomaly detection, overcoming many limitations of traditional methods.

The advancement of artificial intelligence (AI) systems has been propelled by the emergence of large language models (LLMs) and fundamental models. LLMs, being extensive pre-trained statistical language models rooted in neural networks, mainly encompass Transformer-based neural language models pre-trained in vast textual datasets [5].

LLMs, characterized by their expansive scale and pre-trained statistical architectures rooted in neural networks, represent a pivotal development in the AI landscape. These models predominantly encompass Transformer-based neural language models, which undergo extensive pretraining on vast corpora of textual data. Beyond their formidable language comprehension and generation capabilities, LLMs exhibit robust analytical prowess, particularly evident in their adept handling of zero-shot and few-shot data across diverse domains. This attribute underscores the versatility and adaptability of LLMs, positioning them as powerful tools to tackle multifaceted challenges in AI applications. Large language models demonstrate notable proficiency in leveraging contextual information from unlabeled data to infer potential patterns and relationships. This capability is particularly evident in the context of zero-shot Log Anomaly Detection based on Large Language Models and few-shot training across diverse domains, where these models excel in completing corresponding tasks effectively.

Recent advances have focused on compressing long prompt contexts into concise soft prompts to assist LLMs in processing lengthy contextual knowledge more effectively. This approach effectively transforms the original lengthy prompt into a series of manageable short-length soft prompt tokens. Compression-based soft prompts are designed to preserve semantic integrity through self-information [6], instruction fine-tuning [7,8], and performance alignment via knowledge distillation [9,10]. One of the key challenges faced by large language models when processing time-series data is the significant decrease in processing speed as text length increases. Specifically, the attention mechanism in these models exhibits quadratic computational complexity, $O(n^2)$, where n is the text length. As text length grows, both computational demands and processing time increase at a quadratic rate. Additionally, handling long texts exacerbates computational complexity, raises memory and storage demands, and reduces efficiency, which can hinder practical applicability [11,12]. Addressing these issues is essential to optimizing model performance and resource utilization in time-series data processing.

Building on the success of the logPrompt method in online log parsing [13], this paper introduces LLM-Distil, an innovative architecture designed to tackle the performance bottlenecks and computational challenges in processing lengthy text sequences with large language models. The architecture incorporates advanced text compression techniques to improve processing efficiency without compromising anomaly detection accuracy. The process begins with log text preprocessing, where a templating method effectively eliminates redundant information while preserving essential structures, significantly reducing input length. Fine-tuned large language models then further compress the templated text, isolating critical information needed for precise anomaly detection. This two-step compression strategy mitigates the quadratic complexity $O(n^2)$ inherent in the attention mechanism and substantially reduces memory usage and computational demands. Experimental results demonstrate that LLM-Distil achieves anomaly detection accuracy comparable to that of uncompressed text, while significantly reducing processing time and resource consumption. The LLM-Distil framework offers a groundbreaking approach handling large-scale log data, providing significant potential for real-time, resource efficient applications.

II. Related work

Currently, deep learning-based approaches dominate log anomaly detection, taking advantage of the capacity to discern internal relationships within log event sequences. Supervised approaches, such as LogRobust [14] and SwissLog [15], effectively leverage historical log data but require large annotated datasets. This dependency presents significant challenges due to the diverse and dynamic nature of log data. In contrast, unsupervised techniques, such as DeepLog [2] and LogAnomaly [4], which are trained on normal logs, face difficulties in detecting anomalies involving previously unseen log events. Although self-supervised methods like LogBert [16] outperform DeepLog and LogAnomaly, their focus on individual log events often neglects broader contextual relationships, which may lead to missed anomalies. The LogPrompt method utilizes large language models for anomaly detection and demonstrates strong performance in online log analysis. However, its reliance on ChatGPT-4 entails high computational costs, limiting its practicality for widespread use. This paper evaluates the advantages and limitations of these log anomaly detection methods, highlighting the need for innovative research to overcome current challenges and develop more efficient solutions.

In the domain of prompt compression for large language models (LLMs), current research primarily focuses on transforming prompts into soft prompts. These soft prompts are trainable vectors, fine-tuned alongside a specific LLM, effectively encoding the content of lengthy hard prompts into compact, low-dimensional representations.

The first method applies knowledge distillation to generate soft prompts from hard prompts [27,28]. This approach aims to preserve high-level concepts while ensuring that the generated soft prompts maintain the fluency of the original hard prompts. The second method utilizes the summarization capabilities of LLMs to compress long and complex prompts into concise soft prompts [29]. This process involves segmenting the input prompts into smaller units, sequentially condensing their information, and combining the compressed outputs to form the final soft prompt. Another approach, Gist Token [10], condenses instruction prompts into custom prefix soft prompts by training a virtual soft prompt predictor.

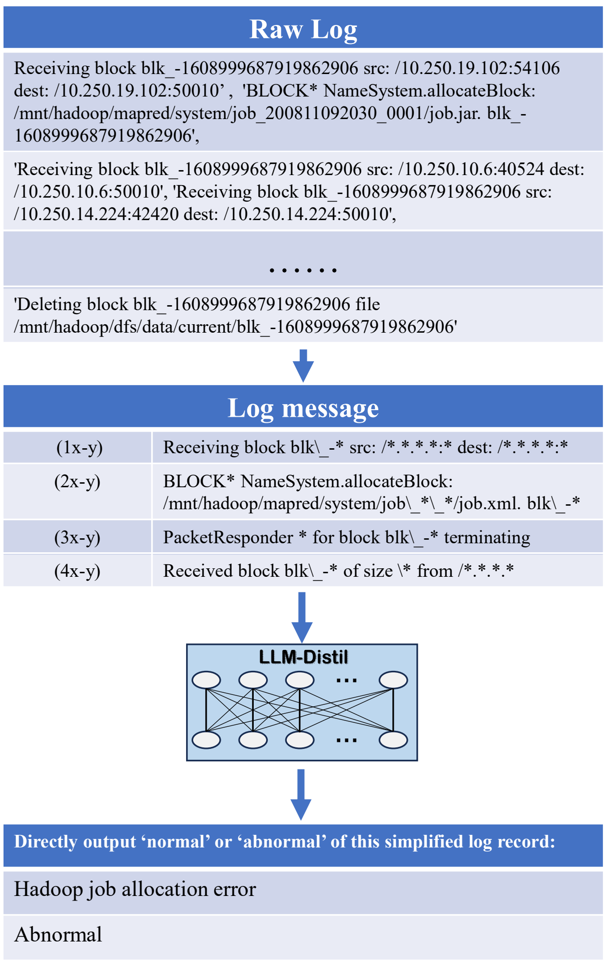
However, the transferability of soft prompt-based compression across various LLMs is limited, requiring the retraining of soft prompts for each change in the specified LLMs. This means that the soft prompts generated are specifically tailored to work only with that particular LLM, which limits their transferability across different LLMs, especially when applied to API-based LLMs.

Prompt compression methods can be categorized into task-aware and task-agnostic approaches based on the use of task information for compression. Task-aware compression reduces context based on the downstream task or current query. For instance, LongLLM Lingua [17] uses a coarse-to-fine compression approach to that is aware of the question to estimate the entropy of the token information, adjusting the estimation based on the question. Reinforcement Learning (RL) methods [18,19] train a model for prompt compression with reward signals from downstream tasks. Soft prompt tuning methods [9,10] typically require fine-tuning for specific tasks. Xu et al. [20] train a summarization model to compress the context based on the question. Task-aware compression approaches are tailored for specific tasks and compression ratios, which may limit their generalizability in real-world applications.

In addition to directly compressing hard prompts into soft prompt vectors, recent advancements [21,22] involve computing the self-information scores or perplexity of the given input context prompt to shorten its length. This process involves filtering out words with lower scores from the input prompt, resulting in a more concise input during inference. The main difference between our work and these recent studies is that they perform prompt compression without considering information from downstream tasks. This leads to inferior performance when directly applied to downstream tasks or transferred between similar but unseen downstream datasets.

III. Method

This paper introduces LLM-Distil, a method for compressing long log texts, consisting of two main steps: log templating, distillation compression. During the training phase, the raw log undergoes preprocessing and templating based on established rules. The processed text, called log messages, is then further compressed through distillation. After obtaining the compressed text, it is essential to maintain a high level of information retention and task performance to ensure that anomaly detection remains comparable to the performance before compression. The overall process framework of LLM-Distil is shown in Fig. 1.



```
Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest: /10.250.19.102:50010' , 'BLOCK* NameSystem.allocateBlock: /mnt/hadoop/mapred/system/job_200811092030_0001/job.jar. blk_-1608999687919862906',
'Receiving block blk_-1608999687919862906 src: /10.250.10.6:40524 dest: /10.250.10.6:50010', 'Receiving block blk_-1608999687919862906 src: /10.250.14.224:42420 dest: /10.250.14.224:50010',
……
'Deleting block blk_-1608999687919862906 file /mnt/hadoop/dfs/data/current/blk_-1608999687919862906'
```

(1x-y)	Receiving block blk_-* src: /*.*.*.*:* dest: /*.*.*.*:*
(2x-y)	BLOCK* NameSystem.allocateBlock: /mnt/hadoop/mapred/system/job_*_*/job.xml. blk_-*
(3x-y)	PacketResponder * for block blk_-* terminating
(4x-y)	Received block blk_-* of size * from /*.*.*.*

Directly output 'normal' or 'abnormal' of this simplified log record:

Hadoop job allocation error

Abnormal

Fig. 1. LLM-Distil implementation overview

A. Log Templating

Drawing inspiration from previous research [23-25], we initially employ the top-K frequent tokens to cluster log messages. The rationale behind this approach is that log messages sharing identical frequent tokens are more likely to possess similar templates. Specifically, we commence by tokenizing each log message and subsequently computing all token frequencies. Throughout this process, we eliminate irrelevant tokens by excluding stop words present in the Scipy library. For every log message, tokens with the highest frequencies are selected, forming the foundation for their classification into various coarse-grained clusters. Essentially, log messages within the same coarse-grained clusters share identical top-K frequent tokens.

However, relying solely on frequent tokens is inadequate for distinguishing log messages with diverse characteristics. For example, log messages sharing the same top-K frequent tokens may correspond to different log templates. To address this limitation, we utilize special characters (e.g., characters excluding alphabets, numerals, or white space) to characterize log messages, defining the set of special characters in a log message as its special format. Log messages stemming from the same template typically exhibit an identical special format. This is because the special characters in the constant parts (e.g., the template) of a log message remain consistent, while those in the dynamic parts (e.g., the parameter) are generally similar. For instance, the special format of "Received block: blk- 160899968 7919862906 of size 6710 from/ 10.250.19.102:54106" consists of [":","-",".","/"], then, replace these parts with wildcard <*>. Similarly, other log messages sharing the same template, such as "Received block: blk-7503483334 202473044 of size 8199 from /10.251.215.16:55695", would have an identical special format. Consequently, we utilize the special formats of log messages to conduct fine-grained clustering. Specifically, log messages within each coarse-grained cluster are further categorized based on their special formats, forming fine-grained clusters where all log messages share not only identical top-K frequent tokens but also the same log format.

The selection of top-K frequent tokens is dataset dependent: for HDFS, K=5 captures template identifiers (e.g., 'Received', 'block', 'size'), while for BGL (unstructured logs), K=8 retains critical system call patterns (e.g., 'ERROR', 'kernel'). Dynamic parameters (e.g., IPv4 addresses formatted as "10.251.215.16" or block IDs like "blk-7503483334") are filtered using a frequency threshold of 0.1% (e.g., tokens appearing in <0.1% of logs are deemed parameters).

After initial processing, many fixed and identifiable parameters remain in the log data. Regular expressions are first applied to eliminate parameters with recognizable patterns. Subsequently, a keyword dictionary is constructed by performing frequency analysis across the entire log dataset. A uniform threshold is set for word frequency, and words exceeding this threshold are included in the initial keyword list. Frequency analysis also generates a list of unique log entries by filtering out duplicates, ensuring that only one instance of each log entry is retained. Each unique log entry is then assigned a distinct identifier. The logs, along with their identifiers, are formatted for clarity and ease of subsequent analysis. This process effectively reduces redundancy, labels the data, and enhances the efficiency of log analysis.

Algorithm 1 Log Template Compression and Integration

```
Require: Log dataset log_col
Ensure: Log template set log_message
1: Parse each log string in log_col into a list of logs
2: for each event log in logs do
3:   Replace all numbers in log with '*' using regular expressions
4: end for
5: Initialize prompt_parts and prompt_parts_count
```

```
6: Set current_paragraph to the prompt header
7: for each log in logs do
8:   if label part of log matches then
9:     Add the label to label_str and append "mal"
10:   end if
11:   if length of current_paragraph plus log exceeds max_len then
12:     Store current_paragraph in prompt_parts
13:     Increment prompt_parts_count
14:     Start a new current_paragraph
15:   end if
16:   Append log to current_paragraph
17: end for
18: Store the final current_paragraph in prompt_parts
19: return log_messag
```

In Algorithm 1, step 3 replaces numeric values (e.g., block IDs and timestamps) with wildcards using regular expressions (e.g., \d+ ⟶ *). This step ensures that dynamic parameters are masked while retaining structural patterns, which is critical for subsequent clustering and compression.

To distinguish static tokens (e.g., log template keywords) from dynamic parameters (e.g., IP addresses, timestamps), we define a frequency-based threshold. Let f(t) denote the occurrence count of token t in the log corpus, and NN be the total number of log entries. A token t is classified as dynamic if its frequency satisfies:

$$f(t) < \gamma \cdot N \tag{1}$$

Here, f(t) is the occurrence count of token t in the log corpus. N is the total number of log entries in the dataset. γ is the threshold for dynamic parameter classification.

B. Distillation Compression

The anomaly detection algorithm identifies key information in the log to determine if it is normal or abnormal. Knowledge distillation, a special knowledge transfer algorithm, migrates knowledge from a larger model to a smaller one, yielding a smaller model with superior performance. This structure is known as a teacher-student network because large models usually perform better than small ones. Enhancing accuracy helps eliminate redundant parameters in large models, improving detection speed and reducing deployment load.

Before the distillation operation, the processed log template is formatted into the input training data. Additionally, an attention mask and position ID must be generated to ensure the model correctly ignores the padded sections during input processing. In detection tasks, we typically use a method that fits the features of the middle layer to train the student network. Specifically, we obtain the significance representation of the output through an affine transformation. The student network is trained through backpropagation to ultimately obtain the student network weights.

The distillation loss consists of two parts: the classification task loss and the Semantic preservation loss. Together, these constitute the overall distillation loss of the network. The calculation process is shown in the figure. The attention mask M, generated by comparing feature divergences between the teacher and student networks, directs the student to prioritize regions of high semantic relevance: for each token pair (i, j), $M_{i,j}$ is computed based on the cosine similarity of their feature representations (Equation. 5), modulated by a temperature parameter σ. This ensures that the student model inherits the teacher's discriminative focus on structural log patterns (e.g., error codes or sequence breaks) while filtering out transient parameters. The attention mechanism is applied in the spatial dimension, and the two are combined to obtain the final attention feature representation.

The LLM-Distil algorithm uses the output features of both the teacher and student networks to obtain the feature attention mask. This mask serves as a weighting coefficient applied to the difference matrix between the teacher and student networks. Finally, the overall distillation loss is used for backpropagation. The specific distillation compression process is shown in Fig. 2.

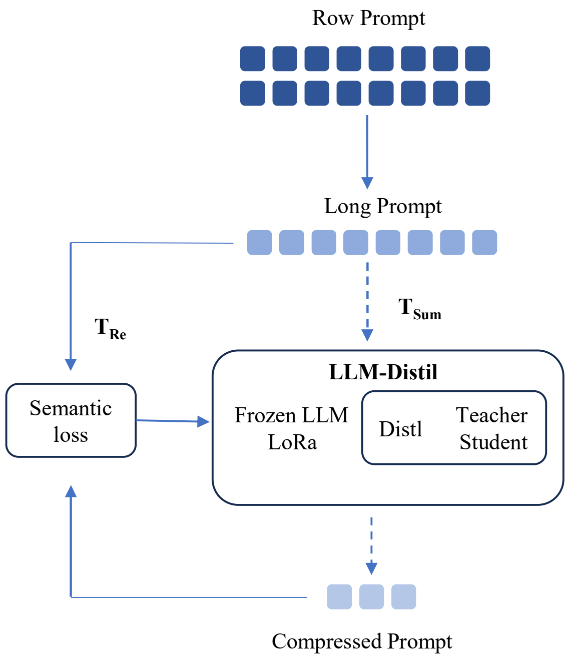


Fig. 2. Distillation compression flowchart

C. Classification Task Loss Function

Prompt compression is formulated as a binary token classification problem, distinguishing between preservation and discarding, to maintain the fidelity of the compressed prompt while ensuring the low latency of the compression model. To extract features and leverage bidirectional contextual information for the token classification model, we use a Transformer encoder. During inference, the decision to retain or discard each token in the original prompt is made based on its probability calculated by our classification model.

Using a Transformer encoder as the feature encoder f_θ, we add a linear classification layer on top. The original prompt, comprising N words $x = \{xi\}^N_{i=1}$, can be formulated as follows:

$$h = f_\theta(x) \tag{2}$$

$$p(x_i, \Theta) = softmax(Wh_i + b) \tag{3}$$

Here, $h = \{hi\}^N_{i=1}$ represents feature vectors for all words, $p(xi, \Theta) \in R^2$ signifies the probability distribution of labels preserve, discard for the i-th word xi, and $\Theta = \{\theta, W, b\}$ denotes all trainable parameters. We denote $y = \{yi\}^N_{i=1}$ as the corresponding labels for all the words in x. The employ cross-entropy loss to train the model. The loss function L concerning x is expressed as follows:

$$L(\Theta) = \frac{1}{N}\sum_{i=1}^{N} CrossEntropy(y_i, p(x_i, \Theta)) \tag{4}$$

Our approach to compressing the original prompt $x = \{xi\}^N_{i=1}$ with a target compression ratio 1/τ involves a three-step process, where τ is defined as the quotient of the number of words in the compressed prompt and the number of words in the original prompt x. First, we derive the target number of tokens to be preserved in the compressed prompt

$\tilde{x}$:Ñ = τN. Next, we use the token classification model to predict the probability p_i of each word x_i being labeled as "preserve". Finally, we retain the top Ñ words in the original prompt x with the highest p_i and maintain their original order to form the compressed prompt $\tilde{x}$.

To further enhance the distillation process, we derive the attention mask M from the feature divergence between the teacher and student networks. Specifically, the attention mask $M_{i,j}$ for the i-th and j-th tokens is computed as follows:

$$M_{i,j} = sigmoid(\frac{\left\| h_i^{teacher} - h_j^{student} \right\|}{\sigma}) \tag{5}$$

where σ is a temperature parameter controlling the sharpness of the mask. This mask is applied to weight the feature differences between the teacher and student networks, ensuring that critical semantic information is preserved during compression.

D. Semantic Preservation Loss Function

We use an unsupervised training approach with semantic preservation loss to compress contexts while retaining their semantic content. We shorten long prompts by summarizing their context and applying our language model loss L_{Sem} to ensure maximal preservation of semantic meaning.

Given the original prompt K = {k_1, … k_n} tokens, and the Compressed Prompt S = {s_1, … s_m} with m tokens, where n ≥ m, our semantic loss aims to ensure maximal preservation of semantics. We measure this by evaluating the similarity between the hidden state embeddings of S and K. To obtain the hidden state embedding of K, we instruct F (· | θ_s) to replicate the input prompt K. This helps in better preserving and embedding the semantic meanings of K. Specifically, d-dimensional hidden state embeddings of K and S can be generated by $e_k \sim$ PF (K | θ_S, T_{Re}) and $e_s \sim$ PF (S | θ_S, T_{Sum}), where T_{Re} and T_{Sum} denote a replicating instruction and a summarizing instruction, respectively. With the aid of T_{Rep}, we instruct F (· | θ_s) to replicate K under the model parameter θ_s, ensuring that $e_K \in R^d$ accurately represents the embedding of K.

The semantic preservation loss L_{Sem} is designed to retain both global semantic alignment and local discriminative power. First, we measure the global similarity between the original prompt K and compressed prompt S using cosine similarity:

$$L_{Sim} = E_S \left[1 - \frac{e_K \cdot e_S}{\|e_K\| \|e_S\|} \right] \tag{6}$$

In this model, we use cosine similarity as a distance function to measure the similarity between e_k and e_S.

To further enhance semantic discrimination, we introduce a contrastive learning loss that distinguishes positive pairs (K, S) from negative samples S′:

$$L_{contrastive} = -log \frac{exp(sim(e_K, e_S) / \tau)}{\sum_{S'} exp(sim(e_K, e_{S'}) / \tau)} \tag{7}$$

Here, sim(·) denotes cosine similarity, S′ represents negative samples, and τ=0.1 is a temperature parameter controlling the distribution sharpness. The total semantic preservation loss combines both components:

$$L_{Sem} = \beta L_{Sim} + (1 - \beta) L_{contrastive} \tag{8}$$

Here, β is the weight coefficient, we set β=0.6 to balance similarity preservation and discriminative power.

The overall distillation loss $L_{Distill}$ integrates the classification task loss L (Equation 4) and the total semantic preservation loss L_{Sem} (Equation 8):

$$L_{Distill} = \alpha L + (1 - \alpha) L_{Sem} \tag{9}$$

Through empirical validation, we set α=0.7 to prioritize detection accuracy while maintaining semantic fidelity.

The two-step compression framework reduces the quadratic complexity of the attention mechanism from $O(n^2)$ to O(nlogn). Let n denote the original token count, and m represent the compressed length (m≪n). This complexity reduction is achieved through a dual mechanism: the templating stage first eliminates transient parameters (e.g., numerical values) to shorten the sequence, and the distillation step further approximates attention operations by prioritizing high-saliency token interactions. Such hierarchical compression mirrors techniques in sparse Transformer architectures [12], where selective token aggregation replaces full self-attention, enabling efficient processing of long sequences. Consequently, the combined steps avoid exhaustive pairwise computations while retaining structural patterns critical for anomaly detection. The total complexity is dominated by O(n′logn′)+$O(m^2)$, resulting in an overall O(nlogn) scaling.

IV. Experimental and Results

In this section, we evaluate the performance of LLM-Distil through experiments aimed at answering three specific research questions:

RQ1: Can anomaly detection in logs achieve the expected results after the log text is compressed using the LLM-Distil method, compared to traditional methods?

RQ2: How effective is the LLM-Distil method in compressing long text?

RQ3: What are the advantages of using compression over directly using large language models for log anomaly detection?

A. Datasets

This paper evaluates the experimental results using real-world log datasets sourced from Loghub [26], namely HDFS, BGL, and Thunderbird. The HDFS dataset encompasses log messages documenting operations and metadata state changes within the Hadoop distributed file system, pivotal for monitoring, troubleshooting, performance tuning, and data analysis. It comprises 11,172,157 log messages, with approximately 284,818 indicating system anomalies. The BGL dataset, recorded by Lawrence Livermore National Laboratory’s BlueGene/L supercomputer system, comprises 4,747,963 log messages, including 348,460 anomalous messages. The Thunderbird dataset, collected from Sandia National Laboratories’ Thunderbird supercomputer, consists of 20,000,000 logs randomly selected for experimentation, incorporating 758,562 abnormal log messages. These datasets serve as robust benchmarks for assessing the efficacy of the proposed log anomaly detection method. Table I shows the statistics of the datasets.

The distribution of datasets used in this study is illustrated in Fig.3. The HDFS, BGL, and Thunderbird datasets exhibit distinct scales in log message volumes (measured in millions, M), anomaly counts (in thousands, K), and sequence classifications. Specifically, HDFS demonstrates a balanced ratio of normal and anomalous sequences, whereas Thunderbird features a significantly higher volume of log messages. The lower subplot further compares matrix values (M/K) and log key distributions across datasets, highlighting Thunderbird's unique characteristic of sparse log keys despite its large-scale log volume. These visualizations collectively emphasize the diversity and complexity of the datasets, ensuring a comprehensive evaluation of the proposed methodology under varied real-world scenarios.

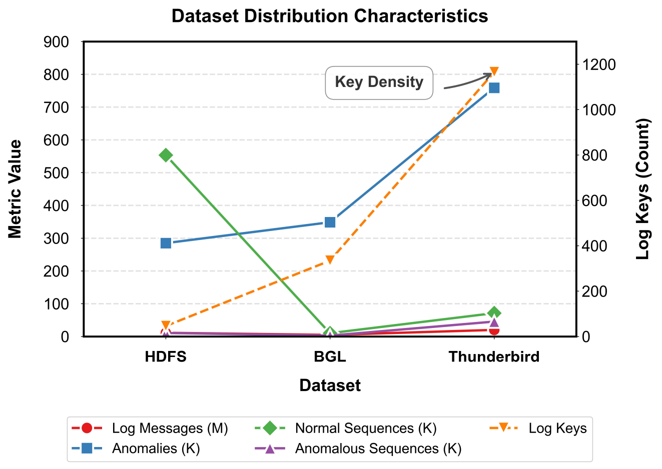


Fig. 3. Dataset distribution across HDFS, BGL, and Thunderbird: log message volumes (M), anomalies (K), normal/anomalous sequences (K), matrix values (M/K), and log key density (sparsity in Thunderbird despite high log volume).

B. Evaluate Metrics

To evaluate the effectiveness of the method, precision, recall and F1 values were used for experimental evaluation, and the specific index calculation formula is as follows:

Precision: The percentage of anomaly log sequences that the model correctly detected out of all detected anomaly log sequences.

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

Recall: The percentage of log data that was correctly detected as an anomaly out of the actual anomaly.

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

F1: A blended average of accuracy and recall to comprehensively evaluate the overall performance of anomaly detection.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{12}$$

In evaluation metrics, TP is the number of abnormal log sequences correctly detected. FP is the number of normal sequences misclassified as anomalies. FN is the number of abnormal sequences misclassified as normal, indicating undetected anomalies.

To verify the effectiveness of text compression, Anomaly Detection Accuracy (ADA) and Information Retention Rate (IRR), are used for evaluation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{13}$$

$$ADA = \frac{Accuracy_{original}}{Accuracy_{compressed}} \tag{14}$$

Here, $Accuracy_{compressed}$ represents the anomaly detection accuracy for the compressed text, while $Accuracy_{original}$ refers to the accuracy for the uncompressed text. An ADA value close to 1 suggests that the compressed text's anomaly detection accuracy is comparable to that of the uncompressed text, implying minimal impact from compression. Conversely, an ADA value significantly lower than 1 indicates that compression has substantially affected detection performance.

IRR measures the proportion of critical information retained in the compressed text compared to the original text. Typically, it is calculated using the BLEU (Bilingual Evaluation Understudy) score, which is a metric commonly employed in machine translation and text generation tasks to assess the similarity between two texts.

$$BLEU = BP \times exp(\sum_{n=1}^{N} w_n log p_n) \tag{15}$$

BP represents the length penalty, which prevents the generation of overly short text. p_n is the precision of the n-gram match, while ω_n is the weight of each n-gram, typically equal for all n-grams. The BLEU score provides a value between 0 and 1, indicating the amount of information retained in the compressed text compared to the original, which is used to calculate the information retention rate.

C. Anomaly Detection Effect(RQ1)

Performance on Log Anomaly Detection. Table II summarizes the performance of LLM-Distil and baseline methods across three datasets. PCA, Isolation Forest, and OCSVM exhibit suboptimal performance in log anomaly detection. Although these methods may achieve high precision or recall individually, they often fail to balance both metrics effectively, leading to suboptimal F1 scores, which are crucial for anomaly detection tasks. This limitation is likely attributed to their reliance on vector-based representations of log sequences, which neglect important temporal patterns.

LogCluster, a method specifically designed for log anomaly detection, outperforms PCA, Isolation Forest, and OCSVM in terms of performance. However, deep learning-based methods, such as DeepLog, LogAnomaly, and LogBert, consistently achieve superior F1 scores, demonstrating their enhanced ability to capture complex log sequence patterns.

To provide a comprehensive visualization of performance distributions across precision, recall, and F1-score metrics, the corresponding comparative analysis is illustrated in Fig. 4.

The LLM-Distil framework compresses log text before anomaly detection using a large language model, retaining essential semantic features while reducing processing time and resource consumption. Experimental results indicate that the LLM-Distil approach not only maintains comparable detection accuracy but often outperforms traditional methods, all while significantly improving computational efficiency.

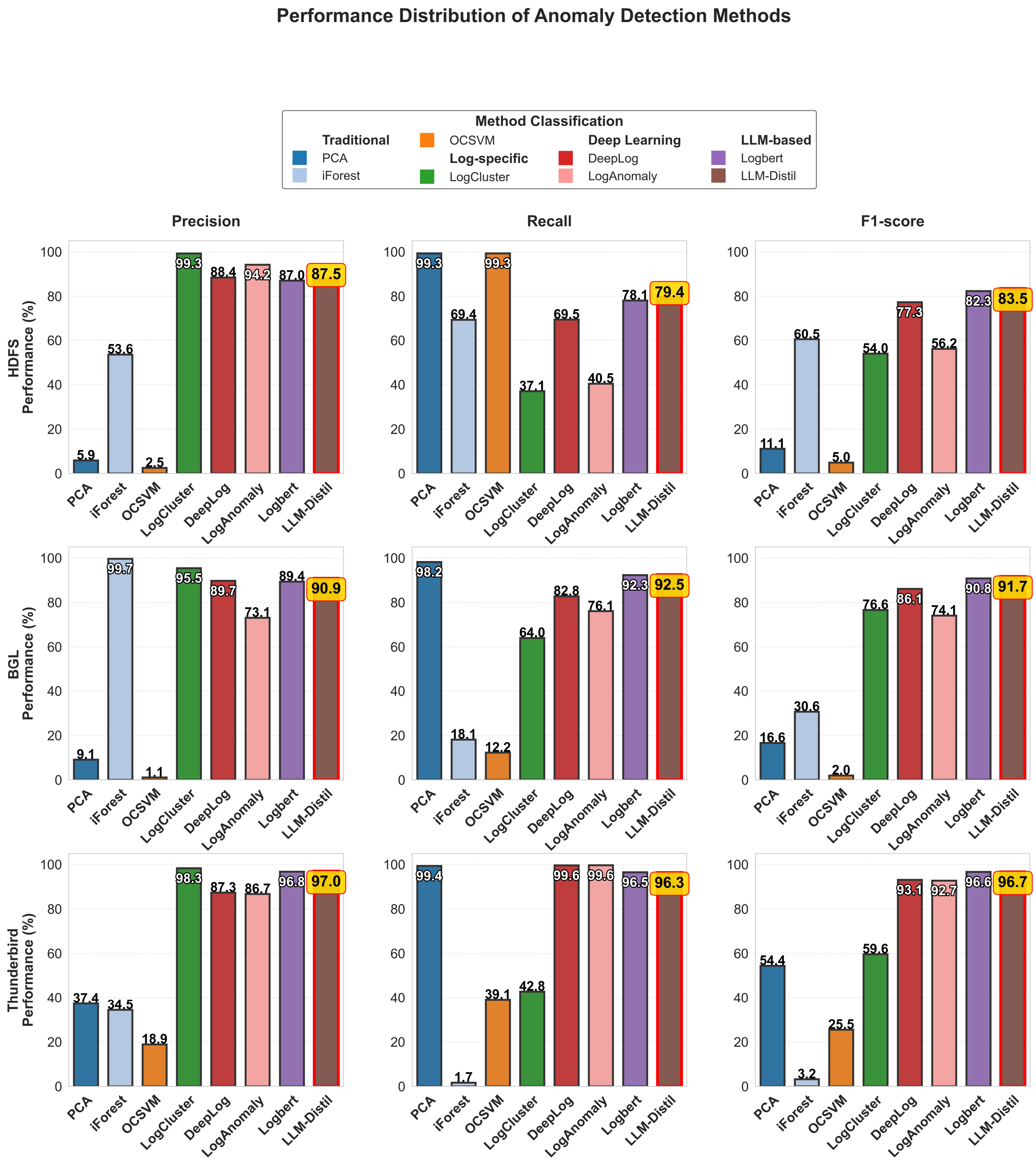


Fig. 4. Performance Distribution of Anomaly Detection Methods across HDFS, BGL, and Thunderbird

D.Ablation Experiments(RQ2)

To verify the effectiveness of our proposed text compression method for processing long texts in large language models, we conducted experiments comparing the performance of compressed and uncompressed text regarding latency, computational resource usage, anomaly detection accuracy, and information retention rate. ChatGLM3 was used as the baseline large language model to evaluate inference efficiency with both raw and compressed text input. In this experiment, we used the same three public datasets as in the anomaly detection experiment: HDFS, BGL, and Thunderbird. We compared our tested compression method against the following three baseline methods:

Baseline-1: The original text, without any preprocessing, is directly fed into the Transformer Encoder for compression.

Baseline-2: Compression is applied after the preprocessing templating step, with no additional fine-tuning.

Baseline-3: Replace the feature extractor from the Transformer Encoder with the LSTM Encoder.

To ensure reproducibility, all experiments were conducted with consistent hyperparameters: a learning rate of 1e-4, a batch size of 32, and 50 training epochs. The statistical significance of performance differences was validated via paired t-tests (α=0.05, df=4). For example, the improvement over Gist Token on HDFS (τ=5%) yielded t=6.21 and p=0.001, confirming the superiority of LLM-Distil under strict compression ratios.

Table III presents the impact of various components on anomaly detection accuracy. Removing the initial templating step (Baseline-1) resulted in a significant reduction in accuracy to 90.3%, with the IRR dropping to 85.73%. This highlights the importance of the templating step in retaining essential information while eliminating redundancy. In the absence of fine-tuning (Baseline-2), both the compression ratio and IRR decreased, leading to a slight drop in anomaly detection accuracy to 94.1%. This underscores the necessity of fine-tuning for optimizing compression and maintaining critical information. Replacing the Transformer Encoder with an LSTM Encoder (Baseline-3) led to a significant decrease in both ADA and IRR, demonstrating that the Transformer Encoder is more effective in capturing global context and retaining key information post-compression. To isolate the impact of model architecture from parameter quantity, we reconfigured the LSTM's hidden layers to match the Transformer's parameter scale. Despite this adjustment, the LSTM variant achieved an F1-score of 89.1% on HDFS, lagging behind the Transformer's 91.7%. This discrepancy highlights the self-attention mechanism's ability to link distant log events. Such capability is vital for identifying multi-step anomalies, which LSTMs struggle to capture due to their limited local receptive field [5]. These findings confirm that the performance drop arises from structural constraints rather than parameter efficiency

In summary, these ablation studies highlight the critical contributions of each component in enhancing the efficiency of text compression and anomaly detection performance. Through optimal integration of these components, the LLM-Distil framework achieves reduced computational overhead and latency while maintaining high information retention.

E. Advantages of Compression(RQ3)

The LLM-Distil algorithm utilizes a large language model for text compression, significantly reducing computational resource consumption while maintaining performance and accuracy.

The LLM-Distil algorithm compresses log text effectively, reducing its length while preserving critical semantic information. The framework's interpretability is further demonstrated through its retention of log templates: for example, a raw log entry containing dynamic parameters (e.g., 'Received block: blk-7503483334202473044 of size 8199 from /10.251.215.16:55695') is transformed into a templated sequence ('Received block: blk-<> *of size* <> from <*>'). By abstracting transient values while highlighting structural patterns, the compressed logs align with human-readable anomaly taxonomies used in manual auditing [14], ensuring that automated detection results remain transparent and actionable.

To comprehensively evaluate the effectiveness of LLM-Distil in log text compression, we compare it against two state-of-the-art prompt compression methods: LongLLM Lingua[17] and Gist Token[10]. The experiments are conducted on the HDFS and Thunderbird datasets under three compression ratios (τ=5%,10%,20%). All methods adopt identical preprocessing steps and test splits to ensure fairness. Key evaluation metrics include ADA (Equation 14), IRR (Equation 15), and F1-score (Equation 12).

As shown in Table IV, LLM-Distil consistently outperforms baseline methods across all compression ratios and datasets. At τ=5%, LLM-Distil achieves 98.5% ADA and 95.28% IRR on HDFS, significantly surpassing LongLLM Lingua (92.1% ADA, 88.9% IRR) and Gist Token (85.4% ADA, 82.3% IRR). Notably, even under extreme compression (τ=5%), LLM-Distil slightly exceeds the F1-score of LogBert (96.67% vs. 96.64%), which processes uncompressed logs. This counterintuitive improvement arises from the framework's ability to both eliminate noise and enhance generalization: the templating stage removes transient parameters that may obscure structural anomalies, while the distillation process enforces the student model to prioritize task-discriminative features inherited from the teacher, thereby suppressing overfitting to redundant tokens [10]. Even under aggressive compression (τ=20%), LLM-Distil maintains robust performance, with 90.1% IRR and 92.6% F1-score on Thunderbird. In contrast, Gist Token suffers from severe over-compression, leading to a 7.3% drop in F1-score on Thunderbird due to truncation of critical dynamic parameters. Fig. 5. illustrates the performance of Gist Token and LLM-Distil under varying compression thresholds across multiple evaluation metrics.

The improvements of LLM-Distil are statistically validated through paired t-tests. On HDFS (τ=5%), the performance gains over LongLLM Lingua and Gist Token are significant (p=0.002 and p=0.001, respectively). Additionally, LLM-Distil reduces inference latency by 31.8% (1500ms vs. 2200ms for Gist Token) and GPU memory usage by 26.1% (6.8 GB vs. 9.2 GB), demonstrating its practical efficiency.

These results highlight the substantial economic benefits of the LLM-Distil algorithm in real-world applications. The algorithm effectively reduces computational overhead, offering significant advantages, particularly in large-scale and high-frequency deployment scenarios.

V. RESULTS

Experimental results demonstrate that the LLM-Distil method effectively retains key semantic information while achieving significant prompt compression. This compression alleviates the challenges posed by long prompts in large language models, such as high computational demands and memory usage, thereby improving processing efficiency and reducing operational costs. The LLM-Distil method reduces the prompt length by 97%, from an average of 1690 tokens to just 50 tokens, and decreases inference delay by a factor of 30, from 2 minutes and 50 seconds to only 10 seconds, while maintaining comparable accuracy and relevance. These results underscore the effectiveness of LLM-Distil for log anomaly detection and further validate the efficiency of compressing long text prompts in large language models.

VI. CONCLUSIONS

Effective log anomaly detection is crucial for identifying and mitigating potential cyberattacks and system failures, ensuring the stability and security of computer systems. This paper introduces LLM-Distil, a novel log anomaly detection framework based on distillation-based compression. The framework first templates the original log sequence, then compresses it to preserve key information. The compressed text is then used for effective anomaly detection. Experimental results on three log datasets show that LLM-Distil achieves anomaly detection accuracy similar to existing methods, while providing substantial improvements in processing speed and computational cost. Compared to conventional approaches, LLM-Distil uniquely integrates template-based preprocessing with knowledge distillation, enabling efficient semantic compression without sacrificing detection accuracy. These findings highlight the potential of LLM-Distil in enhancing the efficiency of large language models for processing long input texts. Although this study focused on log text compression, future work should extend this framework to other sequential domains, particularly investigating how data characteristics impact information retention and model performance. Extending this framework to sequential data domains such as network traffic logs, along with domain-specific adaptations for optimal performance, represents a promising direction. Additionally, further investigation into the model's applicability for anomaly detection across diverse data types is essential for broadening its real-world utility.

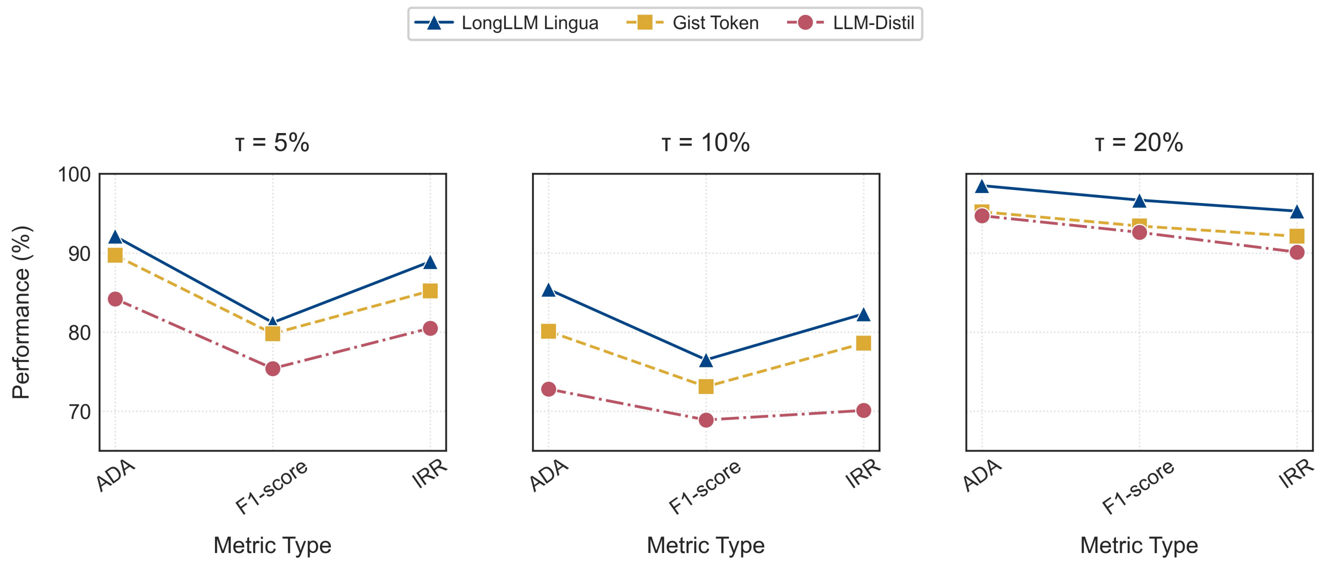


Fig. 5. Performance comparison of Gist Token and LLM-Distil under compression thresholds τ. LLM-Distil exhibits robustness across metrics, particularly at higher compression (τ=20%)

TABLE I
STATISTICS OF EVALUATION DATASETS

Dataset	Log Messages	Anomalies	Log Keys	Log sequences	
				Normal	Anomalous
HDFS	11,172,157	284,818	46	553,366	10,647
BGL	4,747,963	348,460	334	10,045	2,630
Thunderbird	20,000,000	758,562	1,165	71,155	45,385

TABLE II
EXPERIMENTAL RESULTS ON HDFS, BGL, AND THUNDERBIRD DATASETS

Method	HDFS			BGL			Thunderbird		
	Precision	Recall	F-1 score	Precision	Recall	F-1 score	Precision	Recall	F-1 score
PCA	5.89	99.28	11.12	9.07	98.23	16.61	37.35	99.36	54.39
iForest	53.60	69.41	60.49	99.70	18.11	30.65	34.45	1.68	3.20
OCSVM	2.54	99.31	4.95	1.06	12.24	1.96	18.89	39.11	25.48
LogCluster	99.26	37.08	53.99	95.46	64.01	76.63	98.28	42.78	59.61
DeepLog	88.44	69.49	77.34	89.74	82.78	86.12	87.34	99.61	93.08
LogAnomaly	94.15	40.47	56.19	73.12	76.09	74.08	86.72	99.63	92.73
Logbert	87.02	78.10	82.32	89.40	92.32	90.83	96.75	96.52	96.64
LLM-Distil	87.54	79.39	83.45	90.91	92.54	91.72	97.00	96.34	96.67

TABLE III
COMPARISON OF COMPRESSION EFFECT PARAMETERS

	ADA	IRR	Latency(ms)	GPU memory usage(GB)
Baseline-1	90.3%	85.73%	1900	8.3
Baseline-2	94.1%	89.46%	1600	7.5
Baseline-3	92.7%	88.13%	1700	7.9
LLM-Distil	98.5%	95.28%	1500	6.8

TABLE IV
PERFORMANCE COMPARISON WITH PROMPT COMPRESSION BASELINES

Method	τ	ADA	IRR	F1-score	Latency(ms)	GPU memory usage(GB)
LongLLM Lingua	5%	92.1%	88.9%	81.2%	1900	8.3
Gist Token	5%	85.4%	82.3%	76.5%	2200	9.2
LLM-Distil	5%	98.5%	95.28%	96.67%	1500	6.8
LongLLM Lingua	10%	89.7%	85.2%	79.8%	1700	7.5
Gist Token	10%	80.1%	78.6%	73.1%	2000	8.7
LLM-Distil	10%	95.2%	92.1%	93.4%	1300	6.2
LongLLM Lingua	20%	84.2%	80.5%	75.4%	1500	7.0
Gist Token	20%	72.8%	70.1%	68.9%	1800	8.1
LLM-Distil	20%	94.7%	90.1%	92.6%	1100	6.1

REFERENCES

[1] Shimin Tao, Yilun Liu, Weibin Meng, Jingyu Wang, Yanqing Zhao, Chang Su, Weinan Tian, Min Zhang, Hao Yang, and Xun Chen. *Da-parser: A pre-trained domain-aware parsing framework for heterogeneous log analysis.* In 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE, 2023, pp. 322–327.

[2] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. *Deeplog: Anomaly detection and diagnosis from system logs through deep learning*, 2017 pp. 1285–1298.

[3] Zhiwei Wang, Zhengzhang Chen, Jingchao Ni, Hui Liu, Haifeng Chen, and Jiliang Tang. *Multi-scale one-class recurrent neural networks for discrete event sequence anomaly detection*, 2021, pp. 3726–3734.

[4] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. *Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs.*19(7) ,2019, pp. 4739–4745.

[5] Albert Gu, Karan Goel, and Christopher Ré. *Efficiently modeling long sequences with structured state spaces*. arXiv preprint arXiv: 2111.00396, 2021.

[6] Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. *Adapting language models to compress contexts*. arXiv preprint arXiv:2305.14788, 2023.

[7] Tao Ge, Jing Hu, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. arXiv preprint arXiv:2307.06945, 2023.

[8] Siyu Ren, Qi Jia, and Kenny Q Zhu. Context compression for auto-regressive transformers with sentinel tokens. arXiv preprint arXiv:2310.08152, 2023.

[9] David Wingate, Mohammad Shoeybi, and Taylor Sorensen. *Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models.* arXiv preprint arXiv: 2210.03162, 2022.

[10] Jesse Mu, Xiang Li, and Noah Goodman. *Learning to compress prompts with gist tokens*. Advances in Neural Information Processing Systems, 36, 2024.

[11] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Nee- lakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. *Language models are few-shot learners*. Advances in neural information processing systems, 33: 1877–1901, 2020.

[12] Hongye Jin, Xiaotian Han, Jingfeng Yang, Zhimeng Jiang, Zirui Liu, Chia-Yuan Chang, Huiyuan Chen, and Xia Hu. *Llm maybe longlm: Self-extend llm context window without tuning.* arXiv preprint arXiv:2401.01325, 2024.

[13] Yilun Liu, Shimin Tao, Weibin Meng, Jingyu Wang, Wenbing Ma, Yanqing Zhao, Yuhang Chen, Hao Yang, Yanfei Jiang, and Xun Chen. *Logprompt: Prompt engineering towards zero-shot and interpretable log analysis.* arXiv preprint arXiv:2308.07610,2023.

[14] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. *Robust log-based anomaly detection on unstable log data*, 2019, pp. 807–817.

[15] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. *Swisslog: Robust anomaly detection and localization for interleaved unstructured logs.* IEEE Transactions on Dependable and Secure Computing, 2022.

[16] Haixuan Guo, Shuhan Yuan, and Xintao Wu. *Logbert: Log anomaly detection via bert*. pages 1–8,2021.

[17] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. *Longllm-lingua:* Accelerating *and enhancing llms in long context scenarios via prompt compression*. arXiv preprint arXiv:2310.06839, 2023.

[18] .Hoyoun Jung and Kyung-Joong Kim. *Discrete prompt compression with reinforcement learning*. IEEE Access, 2024.

[19] Xijie Huang, Li Lyna Zhang, Kwang-Ting Cheng, and Mao Yang. *Boosting llm reasoning: Push the limits of few-shot learning with reinforced in-context pruning*. arXiv preprint arXiv: 2312.08901, 2023.

[20] Fangyuan Xu, Weijia Shi, and Eunsol Choi. *Recomp: Improving retrieval-augmented lms with context compression and selective augmentation*. In The Twelfth International Conference on Learning Representations, 2023.

[21] Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. *Compressing context to enhance inference efficiency of large language models*. arXiv preprint arXiv:2310.06201,2023.

[22] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. *Llmlingua: Compressing prompts for accelerated inference of large language models*. arXiv preprint arXiv:2310.05736,2023.

[23] Zhihan Jiang, Jinyang Liu, Junjie Huang, Yichen Li, Yintong Huo, Jiazhen Gu, Zhuangbin Chen, Jieming Zhu, and Michael R Lyu. *A large-scale benchmark for log parsing*. arXiv preprint arXiv: 2308.10828,2023.

[24] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R Lyu. *Logzip: Extracting hidden structures via iterative clustering for log compression*, 2019, pp. 863–873.

[25] Meiyappan Nagappan and Mladen A Vouk. *Abstracting log lines to log event types for mining software system logs*, 2010, pp. 114–117.

[26] Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R Lyu. Loghub: *A large collection of system log datasets for ai-driven log analytics*. In 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2023, pp. 355–366.

[27] Wei W, Tang J, Xia L, et al. *Promptmm: Multi-modal knowledge distillation for recommendation with prompt-tuning,* Proceedings of the ACM Web Conference 2024. 2024: 3217-3228.

[28] Zhong Q, Ding L, Liu J, et al. *Panda: Prompt transfer meets knowledge distillation for efficient model adaptation*. IEEE Transactions on Knowledge and Data Engineering, 2024.

[29] Zhang Y, Jin H, Meng D, et al. *A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods*. arXiv preprint arXiv:2403.02901, 2024.