LogFormer: A Pre-train and Tuning Pipeline for Log Anomaly Detection

Hongcheng Guo[1], Jian Yang[1], Jiaheng Liu[1], Jiaqi Bai[1], Boyang Wang[1], Zhoujun Li[1*], Tieqiao Zheng[2], Bo Zhang[2], Junran Peng[1], Qi Tian[3]

[1]State Key Lab of Software Development Environment, Beihang University, Beijing, China
[2]Cloudwise Research, Beijing, China
[3]Huawei, Beijing, China
{hongchengguo,jiaya,liujiaheng,bjq,wangboyang,lizj}@buaa.edu.cn {steven.zheng,bowen.zhang}@cloudwise.com

Abstract

Log anomaly detection is a key component in the field of artificial intelligence for IT operations (AIOps). Considering log data of variant domains, retraining the whole network for unknown domains is inefficient in real industrial scenarios. However, previous deep models merely focused on extracting the semantics of log sequences in the same domain, leading to poor generalization on multi-domain logs. To alleviate this issue, we propose a unified Trans**former**-based framework for **Log** anomaly detection (LogFormer) to improve the generalization ability across different domains, where we establish a two-stage process including the pre-training and adapter-based tuning stage. Specifically, our model is first pre-trained on the source domain to obtain shared semantic knowledge of log data. Then, we transfer such knowledge to the target domain via shared parameters. Besides, the Log-Attention module is proposed to supplement the information ignored by the log-paring. The proposed method is evaluated on three public and one real-world datasets. Experimental results on multiple benchmarks demonstrate the effectiveness of our LogFormer with fewer trainable parameters and lower training costs.

BGL: **Unusual End of Program**
data storage **interrupt**
rts: kernel **terminated** for reason 1004rts: bad message header: [...]
Thunderbird:
kernel: mptscsih: ioc0: attempting task **abort**! (sc=00000101bddee480)
Red Storm:
DMT 310 Command **Aborted**: SCSI cmd:2A LUN 2 DMT 310 T:299 a: [...]

BGL: **Program Not Running**
rts panic! - **stopping execution**
Thunderbird:
pbs mom: Bad file descriptor (9) in tm request, job [job] **not running**
Spirit:
kernel: GM: **LANai is not running**. Allowing port=0 open for debugging
Liberty:
kernel: GM: **LANai is not running**. Allowing port=0 open for debugging

Figure 1: The same anomaly from multiple domains. The top part denotes the "Unusual End of Program" anomaly from three domains including BGL, Thunderbird, and Red Storm while the bottom part is the "Program Not Running" from four domains including BGL, Thunderbird, Spirit, and Liberty.

Introduction

With the rapid development of large-scale IT systems, numerous companies have an increasing demand for high-quality cloud services. Anomaly detection (Breier and Branišová 2015) is critical to monitor data peculiarities for logs, which describe detailed system events at runtime and the intention of users in the large-scale services (Zhang et al. 2015). It is error-prone to detect anomalous logs from a local perspective. In this case, some automatic detection methods based on machine learning are proposed (Xu et al. 2010). Due to the development of IT services, the volume of log data has grown fast and traditional approaches are infeasible. Meanwhile, as log messages are half-structured and have their semantics, it is similar to natural language corpus. Therefore, many deep learning methods based on language models (Hochreiter and Schmidhuber 1997; Devlin et al. 2018) have been proposed on log anomaly detection task (Zhang et al. 2016; Du et al. 2017; Zhang et al. 2019; Meng et al. 2019; Guo et al. 2023a). However, these models adopt parser (Du and Li 2016; He et al. 2017) to gain templates in logs before detection, which leads to the loss of semantics in raw log data.

Despite being different in morphology and syntax, logs of multiple domains usually share similar semantic space. For example, in Fig. 1, three sources (BGL, Thunderbird, Red Storm) have the same anomaly called **Unusual End of Program**. However, existing methods mostly focus on a single domain. When the components from a new domain are introduced, these methods lack the ability to accommodate such unseen logs. Besides, we need to consider the continuous iteration of log data when the system upgrades and it is costly to retrain different models for different datasets.

In this paper, we address the problems above via a two-stage solution called LogFormer. LogFormer is capable of preserving the shared semantic knowledge between different domains. Specifically, to avoid information loss due to parsing, Log-Attention module is to supplement the information ignored by the log-paring for better performance. Then in the first stage, we create a model based on the Log-Attention, which is pre-trained on the source domain to obtain common semantics of log sequences. Second, LogFormer uses a flexible component called Adapter to transfer knowledge from the source domain to the target domain.

*Corresponding author.

Generally, the contributions are as follows: (i) We propose LogFormer, an end-to-end Pre-train and Tuning pipeline to automatically detect log anomalies, which provides a new perspective via simple and effective pre-training and adapter-based tuning strategies for log anomaly detection. (ii) Log-Attention module is proposed to avoid the loss of semantics caused by log parsing. (iii) With only a few additional trainable parameters on the target domain, the training costs are reduced a lot based on the effective parameter-sharing strategy in LogFormer. (iv) LogFormer achieves state-of-the-art performance on three public benchmark datasets.

Related Work

Log Parsing Developers can create an unlimited number of variable names, abbreviations, or special technical terms, which are out of the scale of ordinary English words. If we conduct word splitting, the endless unseen log tokens would explode the vocabulary, which is called the out-of-vocabulary (OOV) problem. To handle this issue, log parsing is used to convert unstructured logs into structured event templates by **keeping keywords** (Jiang et al. 2008; Makanju, Zincir-Heywood, and Milios 2009; He et al. 2017) and **removing extra parameters**, where the parameters usually denote special fields (e.g., /etc, /tmp), words (e.g., *, _), serial numbers (e.g., 0x10001) and so on. In Fig. 2, we use Drain (He et al. 2017) to extract the templates, and each log and the corresponding template are matched. Then, the log template sequence is fed into anomaly detection models.

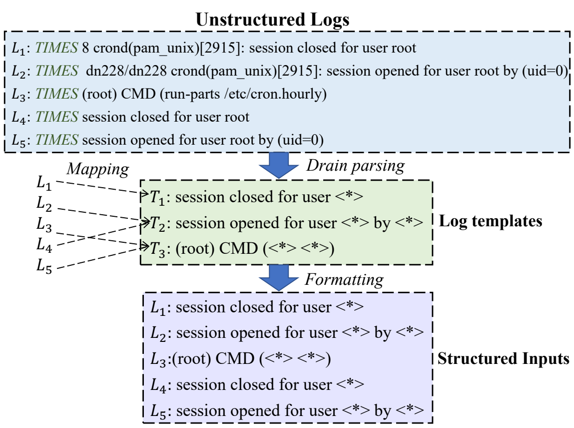

Figure 2: Logs and Templates. The top part is unstructured logs, we adopt Drain algorithm to extract log templates,then we match each log with its template, which is the middle part. The bottom part is structured inputs.

Log Anomaly Detection Natural language processing technology (Bai et al. 2023; Yang et al. 2022; Guo et al. 2022) is evolving rapidly, which has also reinvigorated this task. Supervised methods are based on classification. (Breier and Branišová 2015; Huang et al. 2020; Lu et al. 2018). LogRobust (Zhang et al. 2019) uses both normal and abnormal log data for training based on the Bi-LSTM architecture. Some semi-supervised methods (Xu et al. 2010; Yang et al. 2021b) are proposed to alleviate such burden. DeepLog (Du et al. 2017) uses LSTM to forecast the next log sequence with ranked probabilities. Besides, LogAnomaly (Meng et al. 2019) uses log embeddings to capture the semantic information. PLElog (Yang et al. 2021a) clusters the features of normal data and detects the anomalies by GRU. Although these methods obtain performance improvements on existing log datasets from a single source, they ignore the shared semantics between multiple sources and the value of the parameters removed by log parsing.

Approach

In this section, we describe the general framework of LogFormer. In Fig. 3, LogFormer contains two stages: pre-training and adapter-based tuning. In the following, we present the definition of the problem, and the components of LogFormer. Finally, we introduce the processes of the two stages.

Problem Definition

Log anomaly detection problem is defined as a binary classification task. The model is supposed to determine whether the input log is abnormal or normal. For the source domain, after preprocessing each raw logs, we generate the vector representations of K_{src} log sequences, which are denoted as $S^{src} = \{S_k\}_{k=1}^{K_{src}}$. Then, $S_i^{src} = \{V_t^{src}\}_{t=1}^{T_i^{src}}$ denotes the i-th log sequence, where T_i^{src} is the length of the i-th log sequence and V_t^{src} denotes the t-th log sentence in S_i^{src}. For the target domain, $S^{tgt} = \{S_k^{tgt}\}_{k=1}^{K_{tgt}}$ denotes the representations of K_{tgt} log sequences. $S_j^{tgt} = \{V_t^{tgt}\}_{t=1}^{T_j^{tgt}}$ denotes the j-th log sequence, where T_j^{tgt} is the length of the j-th log sequence. Therefore, the training procedure is as follows. We first pre-train the model on the dataset from source domain as follows:

$$f_p(y_i|S_i^{src}; \Theta)), \tag{1}$$

where f_p represents the pre-training stage and Θ is the model parameters in pre-training stage. Then, the model is transferred to the target domain:

$$f_a(y_j|S_j^{tgt}; \Theta_f, \theta_a). \tag{2}$$

where f_a represents the adapter-based tuning stage. Θ_f denotes the parameter of the encoder transferred from the pre-training stage, which is frozen in adapter-based tuning stage. θ_a is the parameter of the adapter. y is the ground-truth label.

Feature Extractor

The feature extractor converts session sequences (template sequence) to vectors with the same dimension d. Here we use the pre-trained sentence-bert (Reimers and Gurevych 2019) model to get the representation of the template sequence in Fig. 2. Suppose each session has l fixed length, then the embedding of the input X after the feature extractor:

$$X_E = \mathbf{FE}(X). \tag{3}$$

Where **FE** represents the encoder of sentence-bert. we can obtain $X_E \epsilon R^{l \times d}$ for each session.

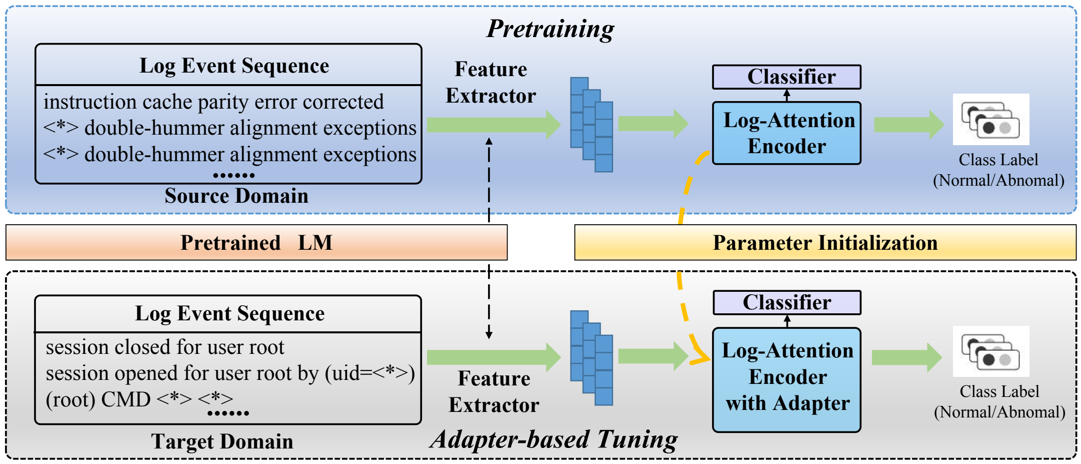

Figure 3: Overview of architecture. Log sequences are first fed into the pre-trained language model to extract features. The Log-Attention encoder is trained on the source domain to acquire shared semantic information. Then, we initialize the encoder and only tune the parameters of the adapter on the target domain to transfer the knowledge.

Log-Attention Module

Although parsing solves the out-of-vocabulary (OOV) problem, the information of the parameters is discarded. To aggregate parameters and keywords information, we have adjusted the structure of the original transformer encoder. Log-Attention module is proposed in Fig. 4. Specifically, after parsing, we gain the P parameters for each log sequence. For each character P_i in P, we adopt the feature extractor to obtain character-level embedding P_i^E. Then we use the Linear layer to encode the whole P^E as follows:

$$\phi_p = \textbf{LINEAR}(P^E). \tag{4}$$

where ϕ_p denotes the output of parameter encoding. Then, we assign each output a learnable scalar, which will serve as a bias term in self-attention. The intuition of Log-Attention is also inspired by position encoding, which is mapped as bias in attention and provides additional position information. The Log-Attention is computed as follows:

$$LogAttention = \textbf{Softmax}(\frac{QK^T}{\sqrt{d/h}} + \phi_p)V. \tag{5}$$

where h is the number of the heads, d denotes the dimension of the input, and Q, K, V represent queries, keys, and values, respectively.

Encoder with Adapter

The order of a log sequence conveys information about the program execution sequence. Wrong execution order is also considered abnormal. Thus, constant positional embedding is also used. The component after the attention layer and feedforward layer is the original serial adapter. We design our log adapter with a parallel structure in Fig. 5, which is inserted parallel to the Log-Attention layer and feedforward layer. This design allows adapter to use input information better with original complete encoders. During adapter-based tuning, only a few parameters of the adapters are updated on the target domain. More specifically, we use down- and up-scale

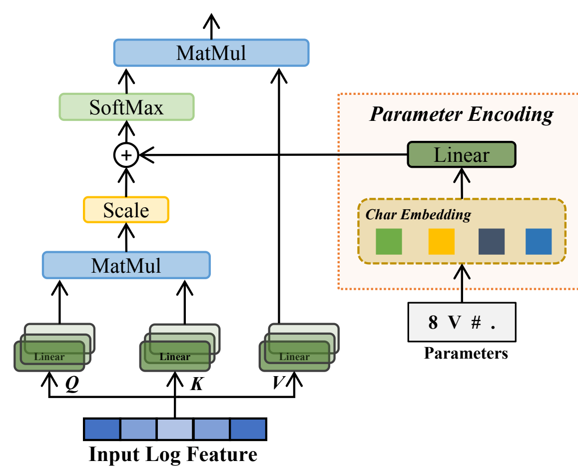

Figure 4: Log-Attention. The left part is the multi-head attention, and the right part is the parameter encoding.

neural networks as the adapter. Two projection layers first map the hidden vector from dimension d to dimension m and then map it back to d. The adapter also has a skip-connection operation internally. The output vector h' is calculated:

$$h' = W_{up} tanh(W_{down} h) + h. \tag{6}$$

where $h \in \mathbb{R}^d$ represents a given hidden vector. $W_{down} \in \mathbb{R}^{m \times d}$ and $W_{up} \in \mathbb{R}^{d \times m}$ is the down-projection and the up-projection matrix respectively, by setting $m << d$, we limit the number of parameters added per adapter, which is the core to reduce trainable parameters while retaining semantic information.

Pre-training

Inspired by pre-trained models (Devlin et al. 2018; Reimers and Gurevych 2019; Guo et al. 2023b), we can acquire the common representation for log anomaly with the stacked log-attention encoders. In this stage, the pre-trained model learns

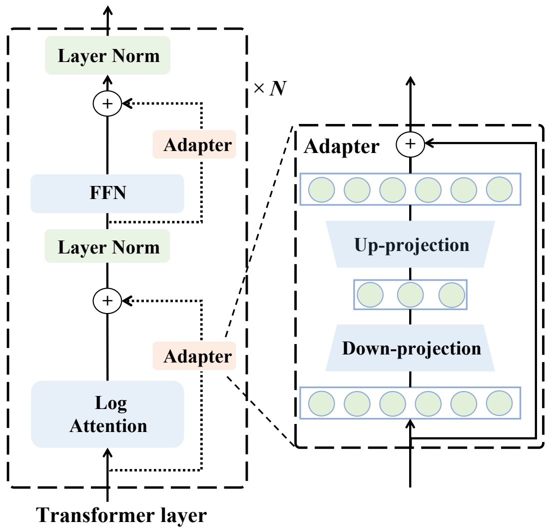

Figure 5: Encoder with Adapters. Where N is the number of encoder layers. The left part describes the log-attention encoder inserted by parallel adapters, and the right part is the structure of an adapter, which is composed of the down- and up-projection layers.

the commonalities among different anomalies. Specifically, the objective of pre-training is a supervised binary classification task without using log adapter. Then, the parameters of the log-attention encoders are used as the initialization of the next stage.

Adapter-based Tuning

Adapter-based tuning leverages the knowledge obtained from the pre-training with lightweight adapters (Houlsby et al. 2019). Specifically, in the second stage, we plug adapters into the encoders of the pre-trained model, where only the parameters of the adapters are updated during target domain adaption. Parameters of the Log-Attention and the feedforward layers in the pre-trained model are frozen. Unlike fine-tuning, LogFormer provides a plug-in mechanism to reuse the pre-trained model with only a few additional trainable parameters.

Training Strategy

The classifier is simply implemented by one linear layer. We both take BCE loss for two stages. Thus, the loss of the pre-training stage is as follows:

$$\mathcal{L}_p = -\mathbb{E}_{x,y \in D_{x,y}^{src}}[\log P(y|x;\Theta)], \tag{7}$$

where $\mathcal{L}_p$ represents the loss in the pre-training stage. Θ is the parameter of the whole model in the pre-training stage. x and y are the input data and label respectively, $D_{x,y}^{src}$ represents the data coming from the source domain. Then, we define the objective loss in the adapter-based tuning stage:

$$\mathcal{L}_a = -\mathbb{E}_{x,y \in D_{x,y}^{tgt}}[\log P(y|x;\Theta_f,\theta_a)]. \tag{8}$$

where $\mathcal{L}_a$ is the loss function in the adapter-based tuning stage. Θ_f is the parameter of the encoder module trained in the pre-training stage, which is frozen in the adapter-based tuning stage. θ_a is the parameter of the adapter. $D_{x,y}^{tgt}$ represents the data coming from the target domain.

Dataset	HDFS	BGL	Thunderbird
Category	Distributed	Supercomputer	Supercomputer
#Messages	11M	5M	10M
#Anomaly	17K	40K	123K
#Templates	49	**1423**	1092
#Error Types	53	**143**	95

Table 1: A summary of the datasets used in this work. Messages are the raw log strings. Log sequences are extracted by ID or sliding window method.

Experiments

In this section, we compare our method with existing methods on multiple benchmark datasets.

Datasets We conduct experiments on three datasets from LogHub (He et al. 2020) [1]. HDFS (Xu et al. 2010) dataset is generated and collected from the Amazon EC2 platform through running Hadoop-based map-reduce jobs. Thunderbird and BGL datasets (Oliner and Stearley 2007) contain logs collected from a two-supercomputer system at Sandia National Labs (SNL) in Albuquerque. The log contains alert and non-alert messages identified by alert category tags. Following (Yao et al. 2020; Meng et al. 2019), 10M/11M/5M continuous log messages from Thunderbird/HDFS/BGL are used.

Preprocessing We extract log sequences by **block IDs** in HDFS. For BGL and Thunderbird, we utilize the **sliding window** (size of 20) without overlap to generate log sequences. We adopt Drain (He et al. 2017) with specifically designed regex to do log parsing. For each dataset, considering that logs evolve over time, we select the first 80% (according to the timestamp of logs) log sequences for training and the rest 20% for testing, which is consistent with the prior work (Yang et al. 2021a; Du et al. 2017).

Implementation Details In experiments, we use different numbers of transformer encoder layers in $\{1, 2, 4\}$. The number of attention heads is 8, and the size of the feedforward network that takes the output of the multi-head self-attention mechanism is 3072. We use Adam as the optimization algorithm whose learning rate is scheduled by OneCycleLR, with $\beta_1 = 0.9$, $\beta_2 = 0.99$, and $\varepsilon = 10^{-8}$. All runs are trained on 2 NVIDIA A100(40G) with a batch size of 64. For each dataset, we tune the maximum learning of the OneCycleLR scheduler in $\{1e-5, 5e-5, 1e-6\}$.

Baselines and Evaluation Table 1 shows five public baselines including Support Vector Machine(SVM), Deeplog (Du et al. 2017), LogAnomaly (Meng et al. 2019), LogRobust (Zhang et al. 2019), and PLELog (Yang et al. 2021a) [2] and

[1] https://github.com/logpai/loghub
[2] https://github.com/YangLin-George/PLELog

Dataset	Method	Precision	Recall	F_1 Score
HDFS	SVM	0.31	0.65	0.41
	DeepLog	0.83	0.87	0.85
	LogAnomaly	0.86	0.89	0.87
	PLELog	0.88	0.93	0.90
	LogRobust	0.88	0.95	0.91
	ChatGPT	0.74	0.82	0.78
	LogFormer$_S$	0.95	0.96	0.95
	LogFormer$_P$	0.96	0.97	0.96
	LogFormer	**0.97**	**0.98**	**0.98**
BGL	SVM	0.22	0.56	0.32
	DeepLog	0.14	0.81	0.24
	LogAnomaly	0.19	0.78	0.31
	PLELog	0.92	0.96	0.94
	LogRobust	0.92	0.96	0.94
	ChatGPT	0.77	0.71	0.74
	LogFormer$_S$	**0.96**	**0.97**	**0.97**
Thunderbird	SVM	0.34	0.91	0.46
	DeepLog	0.48	0.89	0.62
	LogAnomaly	0.51	0.97	0.67
	PLELog	0.85	0.94	0.89
	LogRobust	0.89	0.96	0.92
	ChatGPT	0.84	0.79	0.81
	LogFormer$_S$	0.94	0.98	0.96
	LogFormer$_P$	0.97	0.99	0.98
	LogFormer	**0.99**	**0.99**	**0.99**

Table 2: Results on Thunderbird, BGL and HDFS. LogFormer$_S$ represents the model trained from scratch, LogFormer$_P$ represents the model trained with pre-training but tuning without adapters.

two variants of LogFormer [3]. We also compare with the popular ChatGPT (Ouyang et al. 2022). Here, LogFormer$_S$ is trained from scratch without two stages. LogFormer$_P$ is trained only with the pre-training stage, which means we directly tune the whole parameters from the pre-trained model. For a fair comparison, these baselines are trained on the union of source and target domain, as LogFormer utilize the knowledge of source domain (BGL) and target domain (HDFS/Thunderbird). For evaluation, we use Precision ($\frac{TP}{TP+FP}$), Recall ($\frac{TP}{TP+FN}$) and F_1 score ($\frac{2*Precision*Recall}{Precision+Recall}$).

Main Results In Table 2, baselines are trained on the union of source and target domain data for a fair comparison. In our setting, BGL dataset is chosen as the source domain for the pre-training, HDFS and Thunderbird are chosen as the target domain. LogFormer achieves the highest F_1 score on all three settings. Specifically, results show that most baseline methods perform badly when BGL data is used for training. It is reasonable for the diverse types of error and complex structure of logs in BGL. This also confirms that these baselines have poor generalizability and cannot handle multi-source logs together. When only BGL data is used for training, LogRobust and PLElog achieve a comparable F_1 score with LogFormer$_s$, this means our backbone model with Log-Attention module is strong enough without pre-training and adapter-tuning on the single source.

[3] https://github.com/HC-Guo/LogFormer.

Method	HDFS		BGL		Thunderbird	
	Train	Test	Train	Test	Train	Test
DeepLog	50m	10m	23m	6m	59m	12m
LogAnomaly	1h 48m	22m	1h 10m	20m	1h 43m	30m
PLElog	42m	35s	20m	14s	36m	31s
LogRobust	1h 01m	17m	40m	4m	58m	12m
LogFormer	**29m**	**20s**	**17m**	**11s**	**31m**	**16s**

Table 3: Time consumption of different approaches. The lowest results are highlighted.

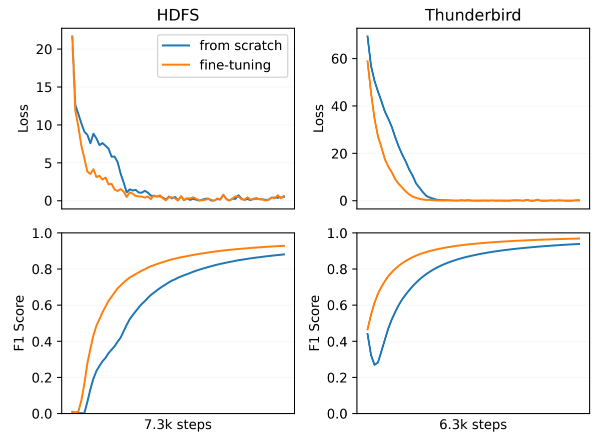

Figure 6: Loss and F_1 score on the test set. We compare two ways of training including training from scratch and fine-tuning from the pre-trained model on BGL.

Time Consumption Table 3 shows the training and testing time of LogFormer on HDFS, BGL, and Thunderbird, respectively. LogFormer gains the lowest training and testing time consumption compared with these state-of-the-art methods.

Ablation Study

Effect of pre-training To demonstrate the effectiveness of pre-training, we compare the performance of LogFormer$_s$ and LogFormer$_p$. We choose BGL as the source domain as its variety in log templates. We compare two strategies in terms of loss and F_1 score. Fig 6 shows the loss and F_1 score curves in the training process (i.e., training steps). Results show that fine-tuning converges faster than training from scratch, which shows the learned knowledge from the source domain is valuable. Besides, the method with fine-tuning achieves higher performance in the initial stage and the loss curve is more stable, which also shows the power of pre-training. For the F_1, we observe that fine-tuning requires fewer training steps to gain the best results, which is noteworthy for reducing costs in industrial scenes. To sum up, the pre-training stage is valuable and allows the model to converge quickly with better results.

Effect of Adapter-based Tuning Although we have shown that pre-training could accelerate convergence without de-

Method	#Layers	#Parameters	HDFS	Thunderbird
Tuning	1	7.2M	0.945	0.969
	2	14.3M	0.962	0.981
	4	28.5M	0.961	0.980
Adapter tuning	1	**0.4M**	0.957	0.972
	2	**0.6M**	0.974	0.987
	4	**1M**	**0.981**	**0.998**

Table 4: Results between fine-tuning and adapter-based tuning. #Layers is the number of encoder layers. #Parameters is the number of trainable parameters.

Attention	HDFS	BGL	Thunderbird
Self-Attention	0.911	0.943	0.935
Log-Attention	**0.952**	**0.974**	**0.962**

Table 5: F_1 scores between self-attention and Log-Attention. Experiments are based on 4 encoder layers.

creasing performance, fine-tuning is expensive and important. Thus, we adopt adapter-based tuning to acquire a compact model for log anomaly detection by adding a few additional trainable parameters. To show the effect of adapter-based tuning, we compare the performance of LogFormer$_p$ and LogFormer on the HDFS and Thunderbird datasets as shown in Table 4. We have the following observations. First, LogFormer generates a little higher F_1 score (1% on average) than directly fine-tuning the pre-trained model on two datasets. Second, Adapter-based tuning adopts $3.5\% - 5.5\%$ of the trainable parameters compared to direct fine-tuning. Third, more encoder layers for fine-tuning do not generate better results. In contrast, adapter-based tuning performs more robustly with more encoder layers.

Effect of Log-Attention We compare the results of the original self-attention with our Log-Attention. To avoid the interference of other factors, we use LogFormer$_S$ in Table. 5. Results show that our Log-Attention achieves 3.6% higher points on average than the self-attention on three datasets, which shows the effect of Log-Attention module. Meanwhile, it shows that variables (removed by log parsing) also provide valuable information for the anomaly detection.

Effect of Variants of Adapters We compare the F_1 scores of the variants of adapters including LoRA (Hu et al. 2022) and Parallel-Adapter (Zhu et al. 2021) in Table 6. Results show that in our task, all three types of adapter gains great performance on three datasets, demonstrating the effectiveness of adapter-based tuning stage.

Effect on Low-resource Setting To verify the power of LogFormer under the low-resource setting, we consider the task with fewer than 20k training examples as the low-resource setting. The ablation study is conducted on the Thunderbird and models are sufficiently trained for 30 epochs. In Fig. 7, we compare the F_1 scores with different numbers of training samples ranging from 5k-20k. We find that 1) Adapter-based tuning consistently outperforms training and fine-tuning, especially when the training size is small. For example, we gain 34% improvements compared with training from scratch with only 5k data. 2) With the number of training samples increasing, the gap between the F_1 scores of all methods will become smaller. 3) LogFormer is robust, with a similar standard deviation across different training sizes. To summarize, LogFormer provides acceptable results in the low-resource setting, which is highly parameter-efficient for log analysis.

Adapter	HDFS	BGL	Thunderbird
Serial Adapter	0.982	0.971	0.992
Parallel Adapter	0.980	0.969	0.988
LoRA	0.981	0.972	0.993

Table 6: F_1 scores between serial adapter and ours. Experiments are based on *4* layers of log-attention encoder.

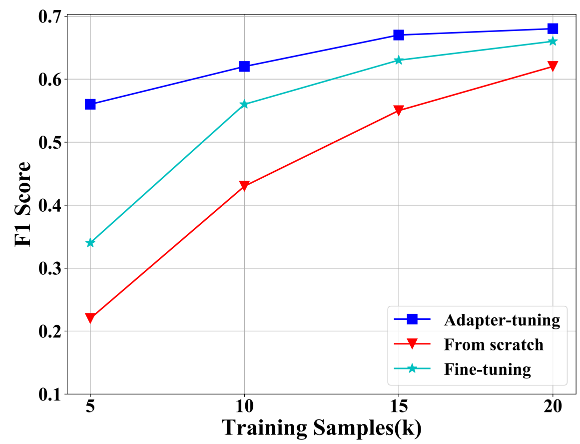

Figure 7: Test performance on Thunderbird w.r.t.the number of training examples. 5k, 10k, 15k, 20k corresponding to the first 1.25%, 2.5%, 3.75%, 5% training data respectively. We show F_1 scores for all methods.

Effect of Source Domain LogFormer aims at transferring knowledge between domains to help detect log anomalies. Thus it is vital to choose the correct source domain for the pre-training stage. A suitable domain needs to meet two conditions: 1) Variety in templates and types of error. 2) For different and similar domains, it has the great power to migrate semantic knowledge. HDFS has fewer templates and types of error compared with BGL and Thunderbird. Thus we do not utilize HDFS as the source domain. Specifically, we compare the results by choosing BGL and Thunderbird as the source domain respectively. In terms of the F_1 score, both of them gain high results on target domains. Thus we turn our attention to loss curves, Fig. 8 shows the loss curves on the target domains. Comparing two pre-trained models, on the HDFS dataset, the model pre-trained on BGL brings faster convergence. Besides, the model pre-trained on BGL brings faster convergence for Thunderbird than Thunderbird brings to BGL. Overall, BGL is the most suitable source domain for

Method	HDFS	BGL	Thunderbird
ChatGPT w/o LCoT	0.78	0.74	0.81
ChatGPT w/ LCoT	0.85	0.83	0.88
LogFormer	**0.98**	**0.97**	**0.99**

Table 7: F_1 between ChatGPT with LCoT and without LCoT.

transferring semantics across domains.

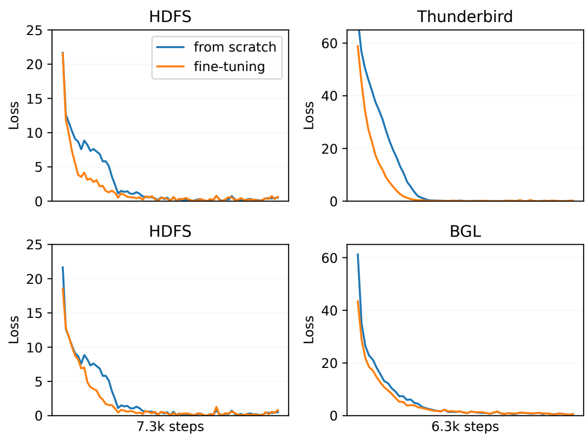

Figure 8: Loss on the test set w.r.t training steps. The upper/bottom results are based on models pre-trained on BGL/Thunderbird. All results are based on using one layer encoder.

Effect of Large Language Models To fully explore the ability of large language models to detect anomalous logs, we specially design a two-hop log Chain-of-Thought (Wei et al. 2022) (LCoT) approach for this task rather than directly generate the decision. Specifically, in Fig. 9, we first guide the model to generate the templates of the log sequences, than we concatenate the raw logs and templates as the prompts, and then let the model determine if there are any anomalies that arise. A comparison of the results using Chain-of-Thought and those generated directly is in Table. 7. We can see that large language models like ChatGPT can not perform well on log anomaly detection task. To explore why the large language model performs badly, we check the results and find that in most bad cases, the variables play an important role. Large language model is more concerned with sensitive words like "Error" and "failed". Such words contain obvious abnormalities, which can be recognized well. The findings further illustrate the importance of our log-attention, and in the future, we can inject external rules to make the large language models more sensitive to the variables.

Practical Evaluation

LogFormer has been successfully applied to a cloud service company. To test the generalization of LogFormer, we conduct experiments on a real-world distributed dataset called GAIA [4]. As the online system serves hundreds of corporations, the generated logs are complex. Thus, it is difficult to detect anomalies on such multi-domain and continuously evolved data. Here we take 8,200,000 log messages for the experiment (80% for training, 20% for testing), amounting to 31,279 anomalous messages. In Table 8, LogFormer still achieves the best performance among these baselines. Besides, LogFormer is stably running over 3000 hours on this system, which further demonstrates the stability of the model.

	Log Context
log sequences:	2023-08-02 10:30:00 DEBUG: Checking server availability. 2023-08-02 10:30:15 ERROR: NetworkException - Unable to establish connection to server. 2023-08-02 10:31:00 INFO: Retrying connection to server.
	Hop 1: Log Parsing
One-hop Prompt:	Extract the templates of *log sequences* while replacing the *variables*with $< * >$
Templates:	1. $< * >$ ERROR: NetworkException - $< * >$ to establish connection to server. 2. $< * >$ DEBUG: Checking server availability. 3. $< * >$ INFO: Retrying connection to server.
	Hop 2: Anomaly Detection
Two-hop Prompt:	According to the *log sequences*, *Templates:*, the relationship between *Templates:* and *variables*, determine if there are any exceptions in templates and variables, and directly give the answer: Yes or No.
Answer:	Yes or No.

Figure 9: An example of two hop log Chain-of-Thought process. We first extract the templates of the logs, then we let the ChatGPT find the anomalies according to the logs and templates.

Dataset	Method	Precision	Recall	F_1 Score
GAIA	SVM	0.21	0.54	0.30
	DeepLog	0.18	0.82	0.31
	LogAnomaly	0.23	0.80	0.36
	PLELog	0.81	0.86	0.84
	LogRobust	0.83	0.94	0.88
	ChatGPT	0.68	0.75	0.71
	LogFormer	**0.89**	**0.98**	**0.93**

Table 8: Results of different methods on GAIA.

Conclusions

In this paper, we propose LogFormer, a pre-train and tuning pipeline for log anomaly detection, which contains the pre-training stage and the adapter-based tuning stage. Besides, the Log-Attention module is proposed to better encode the information of parameters. Extensive experiments show that our LogFormer, with fewer trainable parameters and lower training costs, outperforms all previous baselines.

[4]https://github.com/CloudWise-OpenSource/GAIA-DataSet

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 62276017, U1636211, 61672081), and the Fund of the State Key Laboratory of Software Development Environment (Grant No. SKLSDE-2021ZX-18).

References

Bai, J.; Guo, H.; Liu, J.; Yang, J.; Liang, X.; Yan, Z.; and Li, Z. 2023. GripRank: Bridging the Gap between Retrieval and Generation via the Generative Knowledge Improved Passage Ranking. In *CIKM 2023*, 36–46. ACM.

Breier, J.; and Branišová, J. 2015. Anomaly detection from log files using data mining techniques. In *Information Science and Applications*. Springer.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL 2019*.

Du, M.; and Li, F. 2016. Spell: Streaming parsing of system event logs. In *ICDM 2016*.

Du, M.; Li, F.; Zheng, G.; and Srikumar, V. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *CCS 2017*.

Guo, H.; Guo, Y.; Yang, J.; Liu, J.; Li, Z.; Zheng, T.; Zheng, L.; Hou, W.; and Zhang, B. 2023a. LogLG: Weakly Supervised Log Anomaly Detection via Log-Event Graph Construction. In *DASFAA 2023*, volume 13946 of *Lecture Notes in Computer Science*, 490–501. Springer.

Guo, H.; Liu, J.; Huang, H.; Yang, J.; Li, Z.; Zhang, D.; and Cui, Z. 2022. LVP-M3: Language-aware Visual Prompt for Multilingual Multimodal Machine Translation. In *EMNLP 2022*, 2862–2872. Association for Computational Linguistics.

Guo, H.; Yang, J.; Liu, J.; Yang, L.; Chai, L.; Bai, J.; Peng, J.; Hu, X.; Chen, C.; Zhang, D.; Shi, X.; Zheng, T.; Zheng, L.; Zhang, B.; Xu, K.; and Li, Z. 2023b. OWL: A Large Language Model for IT Operations. *CoRR*, abs/2309.09298.

He, P.; Zhu, J.; Zheng, Z.; and Lyu, M. R. 2017. Drain: An online log parsing approach with fixed depth tree. In *ICWS 2017*, 33–40.

He, S.; Zhu, J.; He, P.; and Lyu, M. R. 2020. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics. *CoRR*, abs/2008.06448.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 1735–1780.

Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; and Gelly, S. 2019. Parameter-efficient transfer learning for NLP. In *ICML 2019*.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *ICLR 2022*.

Huang, S.; Liu, Y.; Fung, C.; He, R.; Zhao, Y.; Yang, H.; and Luan, Z. 2020. HitAnomaly: Hierarchical Transformers for Anomaly Detection in System Log. *TNSM*, 17(4): 2064–2076.

Jiang, Z. M.; Hassan, A. E.; Flora, P.; and Hamann, G. 2008. Abstracting Execution Logs to Execution Events for Enterprise Applications (Short Paper). In *QSIC 2008*, 181–186.

Lu, S.; Wei, X.; Li, Y.; and Wang, L. 2018. Detecting Anomaly in Big Data System Logs Using Convolutional Neural Network. In *DASC 2018*, 151–158.

Makanju, A.; Zincir-Heywood, A. N.; and Milios, E. E. 2009. Clustering event logs using iterative partitioning. In *KDD 2009*, 1255–1264.

Meng, W.; Liu, Y.; Zhu, Y.; Zhang, S.; Pei, D.; Liu, Y.; Chen, Y.; Zhang, R.; Tao, S.; Sun, P.; et al. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. In *IJCAI 2019*.

Oliner, A. J.; and Stearley, J. 2007. What Supercomputers Say: A Study of Five System Logs. In *DSN 2007*, 575–584.

Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C. L.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; Schulman, J.; Hilton, J.; Kelton, F.; Miller, L.; Simens, M.; Askell, A.; Welinder, P.; Christiano, P. F.; Leike, J.; and Lowe, R. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP 2019*, 3980–3990.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E. H.; Le, Q. V.; and Zhou, D. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*.

Xu, W.; Huang, L.; Fox, A.; Patterson, D.; and Jordan, M. I. 2010. Detecting large-scale system problems by mining console logs. In *ICML 2010*.

Yang, J.; Yin, Y.; Ma, S.; Zhang, D.; Wu, S.; Guo, H.; Li, Z.; and Wei, F. 2022. UM4: Unified Multilingual Multiple Teacher-Student Model for Zero-Resource Neural Machine Translation. In *IJCAI 2022*, 4454–4460. ijcai.org.

Yang, L.; Chen, J.; Wang, Z.; Wang, W.; Jiang, J.; Dong, X.; and Zhang, W. 2021a. PLELog: Semi-Supervised Log-Based Anomaly Detection via Probabilistic Label Estimation. In *ICSE 2021*, 230–231.

Yang, L.; Chen, J.; Wang, Z.; Wang, W.; Jiang, J.; Dong, X.; and Zhang, W. 2021b. Semi-supervised Log-based Anomaly Detection via Probabilistic Label Estimation. In *ICSE 2021*, 1448–1460.

Yao, K.; Li, H.; Shang, W.; and Hassan, A. E. 2020. A study of the performance of general compressors on log files. *ESE*, 25(5): 3043–3085.

Zhang, K.; Xu, J.; Min, M. R.; Jiang, G.; Pelechrinis, K.; and Zhang, H. 2016. Automated IT system failure prediction: A deep learning approach. In *BigData 2016*.

Zhang, S.; Liu, Y.; Pei, D.; Chen, Y.; Qu, X.; Tao, S.; and Zang, Z. 2015. Rapidand robust impact assessment of software changes in large internet-based services. In *ENET 2015*.

Zhang, X.; Xu, Y.; Lin, Q.; Qiao, B.; Zhang, H.; Dang, Y.; Xie, C.; Yang, X.; Cheng, Q.; Li, Z.; et al. 2019. Robust log-based anomaly detection on unstable log data. In *FSE 2019*.

Zhu, Y.; Feng, J.; Zhao, C.; Wang, M.; and Li, L. 2021. Serial or Parallel? Plug-able Adapter for multilingual machine translation. *CoRR*.