TRANSLOG: A UNIFIED TRANSFORMER-BASED FRAMEWORK FOR LOG ANOMALY DETECTION

Anonymous authors
Paper under double-blind review

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 especially for low-resource domains. 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 (TRANSLOG) to improve the generalization ability across different domains from a new perspective, 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 adapter, designed for log data, is utilized to improve migration efficiency while reducing cost. The proposed method is evaluated on three public datasets and one real-world dataset. Experimental results demonstrate that our simple yet efficient approach, with fewer trainable parameters and lower training costs in the target domain, achieves state-of-the-art performance on all benchmarks.[1].

1 INTRODUCTION

With the rapid development of large-scale IT systems, numerous companies have an increasing demand for high-quality cloud services. Anomaly detection Breier & Branišová (2015) is a critical substage to monitoring 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 to the point where traditional approaches are infeasible. Therefore, many deep learning methods have been proposed on log anomaly detection task Zhang et al. (2016); Du et al. (2017); Zhang et al. (2019); Meng et al. (2019). Meanwhile, as log messages are half-structured and have their semantics, which is similar to natural language corpus, language models like Transformer Vaswani et al. (2017) and BERT Devlin et al. (2018) are leveraged to obtain semantics in logs.

Despite being different in morphology and syntax, logs of multiple domains share the semantic space. For example, in Fig. 1, three sources (BGL, Thunderbird, Red Storm) have the anomaly called **Unusual End of Program**, thus we naturally think if the model can identify the same anomalies in multiple domains. However, existing approaches mostly focus on a single domain, when new components from a different/similar domain are introduced to the system, they lack the ability to accommodate such unseen log messages. In addition, we need to consider the continuous iteration of log data when system upgrades, as it is costly to retrain different copies of the model. Therefore, a method based on transfer learning is required to perform well on logs from multiple domains.

In this paper, we address the problems above via a two-stage solution called TRANSLOG. TRANSLOG is capable of preserving the shared semantic knowledge between different domains. More specifically, we first create a neural network model based on the self-attention mechanism,

[1]We will release the pre-trained models and code

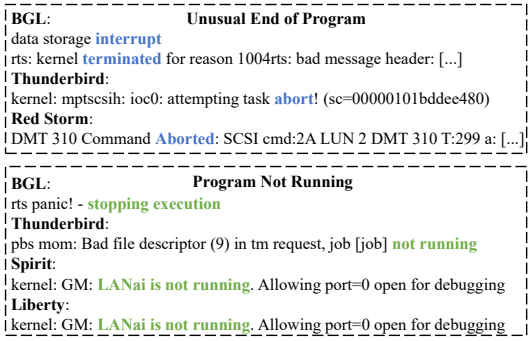

Figure 1: The same anomaly from multiple domains. The top part denotes the **"Unusual End of Program"** anomaly from three domains including BGL, Thunderbird, and Red Storm while the bottom part is the **"Program Not Running"** from four domains including BGL, Thunderbird, Spirit, and Liberty.

which is pre-trained on the source domain to obtain common semantics of log sequences. Second, TRANSLOG utilizes a flexible plugin-in component called adapter to transfer knowledge from the source domain to the target domain.

Generally, the main contributions of this work are listed as follows: (i) We propose TRANSLOG, an end-to-end framework using Transformer encoder to automatically detect log anomalies, which provides a new perspective via simple and effective pretraining and adapter-based tuning strategies for log anomaly detection. (ii) 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 TRANSLOG. (iii) Meanwhile, we design log-adapter with the parallel structure, which outperforms the original serial adapters on benchmark datasets. (iv) TRANSLOG is evaluated on three public and one real-world datasets: HDFS, BGL, Thunderbird, and GAIA and achieves state-of-the-art performance.

2 BACKGROUND

Log Parsing The purpose of log parsing is to convert unstructured log data into the structured event template by removing parameters and keeping keywords Jiang et al. (2008); Makanju et al. (2009); He et al. (2017). In Fig. 2, we utilize Drain to extract all the templates, and then each log message and the corresponding template is matched. Next, the whole log template sequence is fed into anomaly detection models.

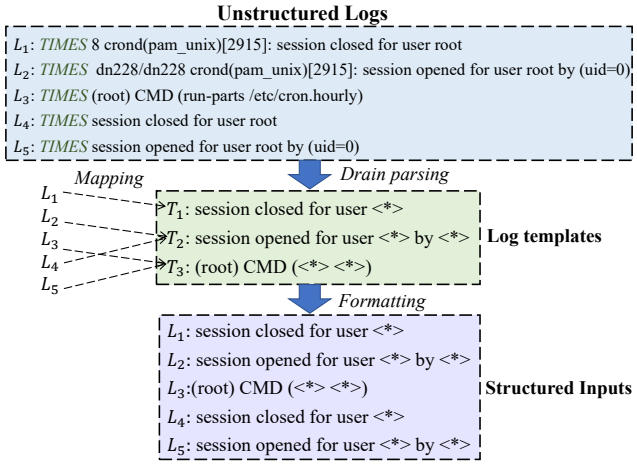

Figure 2: Logs and Templates. The top part is unstructured logs, we adopt Drain algorithm to extract log templates,then we match each log with its template, which is the middle part. The bottom part is structured inputs.

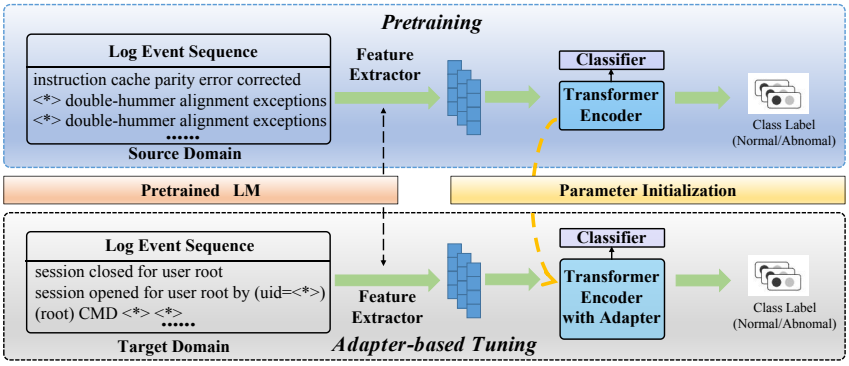

Figure 3: Overview of our proposed architecture. All log event sequence is first fed into the pre-trained language model to extract the representations. The Transformer encoder is trained on the high-resource source-domain dataset to acquire shared semantic information. Then, we initialize the Transformer encoder and only tune the parameters of the adapter on the target-domain dataset to transfer the knowledge from the source domain to the target domain.

Log Anomaly Detection There are two main methods for log anomaly detection, including supervised and unsupervised methods. Supervised methods are often classification-based methods Breier & Branišová (2015); Huang et al. (2020); Lu et al. (2018); Wittkopp et al. (2021b). LogRobust Zhang et al. (2019) utilizes both normal and abnormal log data for training based on the Bi-LSTM architecture. However, obtaining system-specific labeled samples is costly and impractical. Some unsupervised methods Xu et al. (2010); Yang et al. (2021b); Wittkopp et al. (2021a) have been proposed to alleviate such burden. DeepLog Du et al. (2017) utilizes the LSTM network to forecast the next log sequence with the ranked probabilities. Besides, LogAnomaly Meng et al. (2019) utilizes the embeddings of logs to capture the semantic information. PLElog Yang et al. (2021a) is a semi-supervised method to cluster the features of normal data then detect the anomalies by GRU module.

Although these methods attain the improvement of performance on the single log source, they ignore the shared semantics between multiple sources. In contrast, TRANSLOG leverages the semantic knowledge efficiently via the Transformer-adapter architecture.

3 TRANSLOG

In this section, we describe the general framework for log anomaly detection, named TRANSLOG. The architecture of the TRANSLOG is shown in Fig. 3, which contains two stages: pre-training and adapter-based tuning. In the following, we start with the definition of the problem, and then the components of the backbone model are presented. Finally, we introduce the processes of the two stages.

3.1 PROBLEM DEFINITION

Log anomaly detection problem is defined as a dichotomy problem. The model is supposed to determine whether the input log is abnormal or normal. For the source domain, assuming that through preprocessing, we achieve the vector representations of K_{src} log sequences, which is 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. 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 defined as follows. We first pre-train the model on the source-domain dataset:

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

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

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

where f_a represents the adapter-based tuning stage. Θ_f is the parameter of the transformer 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 groundtruth. Through Equation 1 and 2, TRANSLOG learns the semantic representation of template sequences between domains.

3.2 BACKBONE MODEL

Feature Extractor The feature extractor converts session sequences (template sequence) to vectors with the same dimension d. Here we use the pre-trained sentence-bertReimers & Gurevych (2019) model to get the template sequence representation. Recently some methods extract semantic representation from raw log messages, they believe it could prevent the loss of information due to log parsing errors. However, embedding every log message is not realistic considering a large amount of log data. Studies also show that almost all anomalies could be detected by template sequence, even if there are parsing errors. Thus, we only embed all existing log templates. Each session has l fixed length, so through the layer, we can obtain the $X \epsilon R^{l \times d}$ for each session.

3.3 ENCODER WITH LOG-ADAPTER

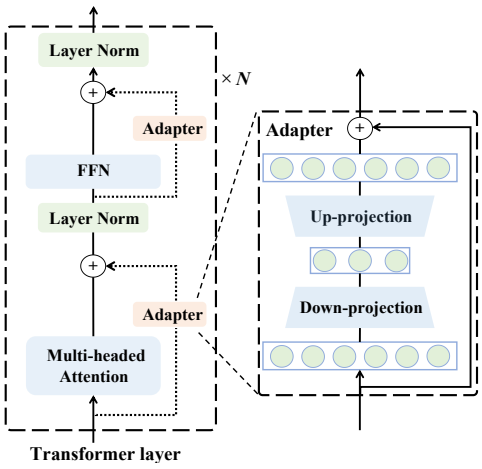

Figure 4: Encoder with Log-Adapter. Where N is the number of transformer layers. The left part describes the transformer encoder inserted by parallel adapters, the right part is the structure of an adapter, which is composed of the down- and up-projection layers.

In Fig. 4, To better encode the corresponding feature of inputs, we use the transformer encoder as the backbone model. By doing so, our encoder with self-attention mechanism overcomes the limitations of RNN-based models. The core self-attention mechanism is formally written as:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d/h}})V. \tag{3}$$

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.

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 self-attention layer and feedforward layer is the original serial adapter. We design our log adapter with a parallel structure, which is inserted parallel to the self-attention layer and feedforward layer. This design allows adapter to utilize input information better with original complete Transformer 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 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' of the adapter is calculated:

$$h' = W_{up} tanh(W_{down} h) + h. \tag{4}$$

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 to the maximum extent.

3.4 PRE-TRAINING

In this stage, the pre-trained model learns the commonalities among different anomalies from the semantic level, which contributes significantly to the anomaly detection for new log sources. More specifically, the objective of this stage is the same as anomaly detection, which is a supervised classification task without adapters in the model. We acquire the common reason for log anomaly with the stacked transformer encoder. Then, the parameters of the transformer encoder, which is trained during this stage, are shared to the next stage.

3.5 ADAPTER-BASED TUNING

When tuning a pre-trained model from the source domain to a target domain, the way of adapter-based tuning leverages the knowledge obtained from the pre-training stage with lightweight adapters, which are neural networks like Houlsby et al. (2019). In this paper, our log adapter is composed of one down-projection layer, one activation layer, and one up-projection layer in Fig. 4. Through the pre-training stage, we achieve the pre-trained model, thus in this second stage, we plug adapters into the transformer layers of the pre-trained model, afterward, only the parameters of the adapters are updated during target domain adaption. Parameters of the multi-headed attention and the feedforward layers in the pre-trained model are frozen. Unlike fine-tuning, TRANSLOG provides a plug-in mechanism to reuse the pre-trained model with only a few additional trainable parameters.

3.6 TRAINING STRATEGY

In this two-classification task, the classifier is simply implemented by one liner layer. We both take BCE loss for two stages. Thus, we define the objective loss of the pre-training stage as follows:

$$\mathcal{L}_p = -\mathbb{E}_{x,y \in D^{src}_{x,y}}[\log P(y|x;\Theta)], \tag{5}$$

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, $D^{src}_{x,y}$ represents the data coming from the source domain. Then, we define the objective loss in the adapter-based tuning stage:

$$\mathcal{L}_a = -\mathbb{E}_{x,y \in D^{tgt}_{x,y}}[\log P(y|x;\Theta_f,\theta_a)]. \tag{6}$$

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. $D^{tgt}_{x,y}$ represents the data coming from the target domain.

4 EXPERIMENTS

In this section, the comprehensive settings of the experiment are illustrated. Compared with baseline methods, our TRANSLOG reaches the state-of-the-art performance on all datasets.

Datasets Experiments are conducted on three public datasets from LogHub He et al. (2020) [2]. 10M/11M/5M continuous log messages from Thunderbird/HDFS/BGL are separately leveraged, which is consistent with the prior work Yao et al. (2020); Le & Zhang (2021). HDFS Xu et al.

[2]https://github.com/logpai/loghub

(2010) is generated and collected from the Amazon EC2 platform through running Hadoop-based map-reduce jobs. Thunderbird and BGL Oliner & 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. Thorough details of datasets are provided in Appendix A

Preprocessing and Implementation Different datasets require preprocessing correspondingly. We extract log sequences by block IDs for HDFS, since logs in HDFS with the same block ID are correlated. BGL and Thunderbird do not have such IDs, so we utilize a sliding window(size of 20) without overlap to generate a log sequence. We adopt Drain He et al. (2017) with specifically designed regex to do log parsing. Number of anomaly is counted based on window. Windows containing anomalous message are considered as anomalies. For each dataset, considering that logs evolve over time, we select the first 80% (according to the timestamps 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). We provide full details about the implementation in Appendix C

Baselines and Evaluation We compare TRANSLOG with the five public baseline methods and two variants of our method in Table 1. Five public methods are 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) [3]. Two variants are TRANSLOG$_S$ and TRANSLOG$_P$, TRANSLOG$_S$ represents that the model is trained from scratch without pre-training and adapter-based tuning stage. TRANSLOG$_P$ represents the model is trained with the pre-training stage but without the adapter-based tuning stage, which means we directly tune the parameters transferred from the pre-trained model. For a fair comparison, these baselines are reproduced on the union of source and target domain data, as TRANSLOG utilize the knowledge of source domain (BGL) and the target domain (HDFS/Thunderbird). In our experiments, we use precision ($\frac{TP}{TP+FP}$), recall ($\frac{TP}{TP+FN}$) and F_1 score ($\frac{2*Precision*Recall}{Precision+Recall}$) to compare our method and previous baselines.

Main Results In Table 1, as TRANSLOG utilizes the knowledge of source and target domain, we train the baseline methods on BGL+HDFS, BGL+Thunderbird, and BGL datasets for fair comparison. TRANSLOG achieves the highest F_1 score on all three settings. SpecificallyTo obtain our main results, BGL is chosen as the source domain for the pre-training, then knowledge is transferred to the target domains (HDFS/Thunderbird) via adapter-based tuning. Results show that most baselines perform badly when BGL data is added to training. It is reasonable for the diverse types of error and complex structure of logs in BGL. This is also the reason why we choose the BGL dataset for pre-training. Although LogRobust achieves a comparable F_1 score with TRANSLOG, our method has less time consumption and fewer trainable parameters. We compare TRANSLOG$_S$, TRANSLOG$_P$ and complete TRANSLOG, the improvement on BGL demonstrates that the effect of pre-training stage and the adapter tuning stage, even the source and target domain are both BGL. We provide a thorough analysis of our model in Section 5.

To test the generalization of TRANSLOG, we conduct the experiment on a real-world distributed online system. TRANSLOG still achieves the best results. Details are provided in Appendix E.

As for the time consumption, TRANSLOG is efficient with short testing time and satisfactory offline training time. Since the training process is offline, the time consumption of TRANSLOG is acceptable. Despite the efficiency of all these methods, our method achieves the least testing time. Detailed results of comparison is provided in Appendix D

5 ABLATION STUDY

Effectiveness of pre-training To demonstrate the feasibility of transferring semantics between domains through pre-training, we compare the performance of two training ways (training from scratch and fine-tuning). Here, the way of fine-tuning is to update the parameter of the whole pre-trained model without adapters.

[3] https://github.com/YangLin-George/PLELog

Table 1: Experimental results compared with baselines on Thunderbird, BGL and HDFS. The best results are highlighted. TRANSLOG$_S$ represents the model trained from scratch, TRANSLOG$_P$ represents the model trained with the pre-training but tuning without adapters. TRANSLOG represents the model with pre-training and adapter-based tuning stages.

Dataset	Method	Precision	Recall	F_1 Score
HDFS	SVM	0.31	0.65	0.41
	DeepLog	0.36	0.87	0.51
	LogAnomaly	0.42	0.96	0.58
	PLELog	0.90	0.92	0.91
	LogRobust	0.95	0.97	0.96
	TRANSLOG$_S$	0.97	0.98	0.97
	TRANSLOG$_P$	0.98	0.99	0.98
	TRANSLOG	**0.99**	**0.99**	**0.99**
BGL	SVM	0.22	0.56	0.32
	DeepLog	0.14	0.81	0.24
	LogAnomaly	0.19	0.78	0.31
	PLELog	0.96	0.98	0.97
	LogRobust	0.96	0.96	0.96
	TRANSLOG$_S$	0.96	0.97	0.96
	TRANSLOG$_P$	0.97	0.98	0.98
	TRANSLOG	**0.99**	**0.99**	**0.99**
Thunderbird	SVM	0.34	0.91	0.46
	DeepLog	0.48	0.89	0.62
	LogAnomaly	0.51	0.97	0.67
	PLELog	0.91	0.95	0.93
	LogRobust	0.94	0.94	0.94
	TRANSLOG$_S$	0.97	0.97	0.97
	TRANSLOG$_P$	0.97	0.99	0.98
	TRANSLOG	**0.99**	**0.99**	**0.99**

We compare two strategies in terms of their rate of convergence and final results. Fig 5 displays the loss curves and F_1 score curves w.r.t training steps. The results show that fine-tuning converges faster than training from scratch, which demonstrates that semantic knowledge from the pre-trained model is valuable. Besides, fine-tuning achieves higher performance than training from scratch at first steps, which shows the power of transfer learning. We further discover that fine-tuning is more stable with a smoother curve on Thunderbird, illustrating that similar domains share semantics more efficiently.

As TRANSLOG achieves good performance even trained from scratch, the final F_1 scores of two methods are close. From the F_1 score, we observe that fine-tuning requires fewer training steps to gain the best result, which is noteworthy for reducing costs in industrial scenes. To sum up, the pre-training stage is valuable and will allow the model to converge quickly with better results.

Effectiveness of Adapter-based Tuning Although we have verified that pre-training can accelerate convergence without reducing performance, fine-tuning each component is expensive and inconvenient. Thus, we adopt the adapter-based tuning with modified log-adapter. By utilizing the adapter, we acquire a compact model for the industry application by adding a few additional trainable parameters.

To confirm the efficiency of the adapter-based tuning, we compare the performance of directly fine-tuning and adapter tuning. Specifically, we conduct an analysis by adjusting the number of encoder layers in $\{1, 2, 4\}$. Table 2 shows that 1) TRANSLOG with log-adapter achieves higher F_1 score (1% on average) than directly fine-tuning on two datasets. 2) Adapter-based tuning adopts $3.5\% - 5.5\%$ of the trainable parameters compared to directly fine-tuning. 3) More encoder layers for fine-tuning do not generate better results. Simultaneously, adapter-based tuning performs more robustly when we stack more encoder layers.

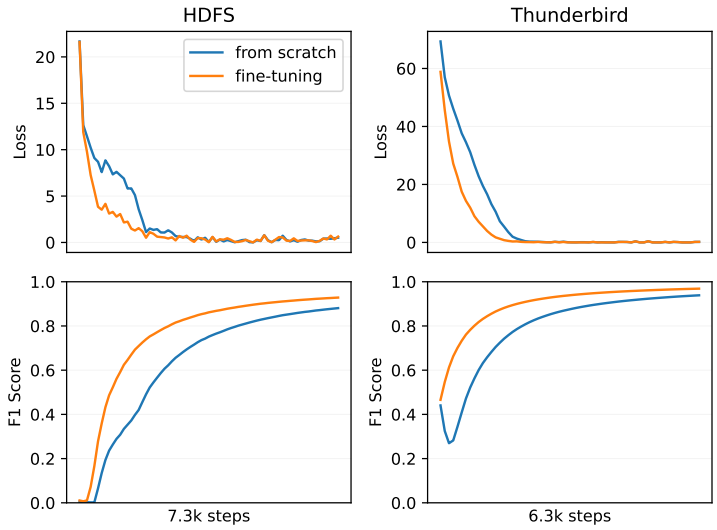

Figure 5: Loss and F_1 score on the test set w.r.t training steps. The left/right part represents Loss/F_1 score of the HDFS/Thunderbird. We compare two ways of training including training from scratch and fine-tuning from model pre-trained on BGL. All results are using 1-layer Transformer encoder and the same learning rate.

To prove the effectiveness of Log-Adapter module, we compare our Log-Adapter with the traditional serial adapter. Full results are provided in Appendix B

Table 2: Results between fine-tuning and adapter-based tuning. *Layers* is the number of encoder layers. *Parameters* is the number of trainable parameters in the model.

Method	Layers	Parameters	HDFS	Thunderbird
Tuning	1	7.2M	0.978	0.982
	2	14.3M	0.982	0.981
	4	28.5M	0.981	0.982
Adapter tuning	1	**0.4M**	0.991	0.990
	2	**0.6M**	0.994	0.996
	4	**1M**	**0.998**	**0.998**

Source Domain Chosen TRANSLOG 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 **0.99** on target domains. Thus we turn our attention to loss curves, Fig. 6 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. In a word, BGL is the most suitable source domain for transferring semantics to similar and dissimilar domains.

Low-resource Setting To verify the power of TRANSLOG 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

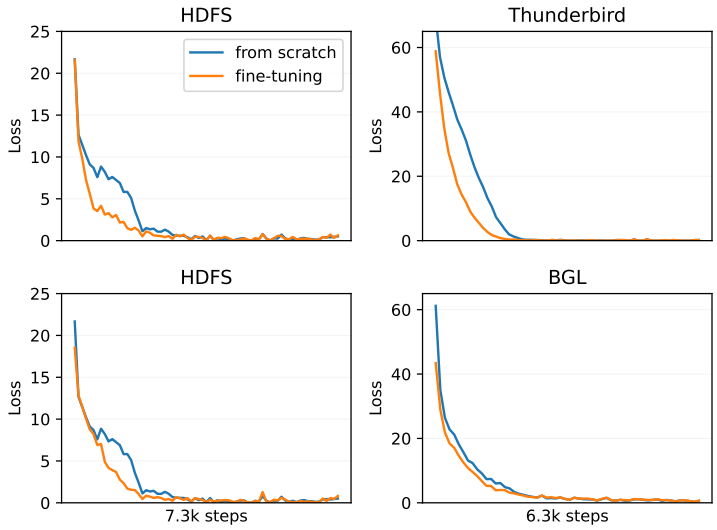

Figure 6: Loss on the test set w.r.t training steps. The upper/bottom results are based on parameters pre-trained on BGL/Thunderbird, thus BGL/Thunderbird are not shown. All results are using 1-layer Transformer encoder and the same learning rate.

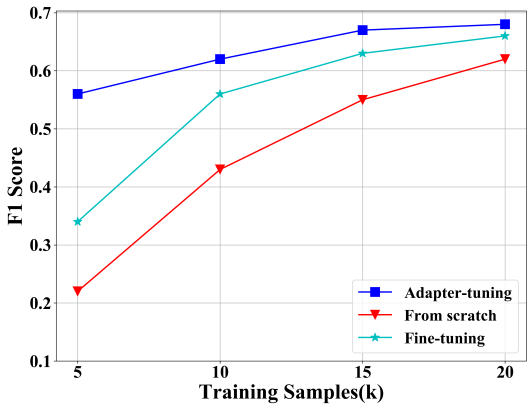

Figure 7: Test performance on Thunderbird w.r.t.the number of training examples. 5k, 10k, 15k, 20k corresponding to the first 1.25%, 2.5%, 3.75%, 5% training data respectively. We show F_1 scores for all methods.

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) TRANSLOG is robust, with a similar standard deviation across different training sizes. To summarize, TRANSLOG provides acceptable results in the low-resource setting, which is highly parameter-efficient for log analysis.

6 CONCLUSION

In this paper, we propose TRANSLOG, a unified transformer-based framework for log anomaly detection, which contains the pre-training stage and the adapter-based tuning stage. Extensive experiments demonstrate that our TRANSLOG, with fewer trainable parameters and lower training costs, outperforms all previous baselines. We foresee the semantic migration between log sources for a unified multiple sources detection.

REFERENCES

Jakub Breier and Jana Branišová. Anomaly detection from log files using data mining techniques. In *Information Science and Applications*. 2015.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL 2019*, 2018.

Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *CCS 2017*, 2017.

Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *ICWS 2017*, pp. 33–40, 2017.

Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. Loghub: A large collection of system log datasets towards automated log analytics. *CoRR*, abs/2008.06448, 2020.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *ICML 2019*, 2019.

Shaohan Huang, Yi Liu, Carol Fung, Rong He, Yining Zhao, Hailong Yang, and Zhongzhi Luan. Hitanomaly: Hierarchical transformers for anomaly detection in system log. *TNSM*, 17(4):2064–2076, 2020.

Zhen Ming Jiang, Ahmed E. Hassan, Parminder Flora, and Gilbert Hamann. Abstracting execution logs to execution events for enterprise applications (short paper). In *QSIC 2008*, pp. 181–186, 2008.

Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection without log parsing. *arXiv preprint arXiv:2108.01955*, abs/2108.01955, 2021.

Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. Detecting anomaly in big data system logs using convolutional neural network. In *DASC 2018*, pp. 151–158, 2018.

Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. Clustering event logs using iterative partitioning. In *KDD 2009*, pp. 1255–1264, 2009.

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

Adam J. Oliner and Jon Stearley. What supercomputers say: A study of five system logs. In *DSN 2007*, pp. 575–584, 2007.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP 2019*, pp. 3980–3990, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NIPS 2017*, 2017.

Thorsten Wittkopp, Alexander Acker, Sasho Nedelkoski, Jasmin Bogatinovski, Dominik Scheinert, Wu Fan, and Odej Kao. A2log: Attentive augmented log anomaly detection. *CoRR*, 2021a.

Thorsten Wittkopp, Philipp Wiesner, Dominik Scheinert, and Alexander Acker. Loglab: Attention-based labeling of log data anomalies via weak supervision. In *ICSOC 2021*, pp. 700–707, 2021b.

Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *ICML 2010*, 2010.

Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. Plelog: Semi-supervised log-based anomaly detection via probabilistic label estimation. In *ICSE 2021*, pp. 230–231, 2021a.

Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *ICSE 2021*, pp. 1448–1460, 2021b.

Kundi Yao, Heng Li, Weiyi Shang, and Ahmed E. Hassan. A study of the performance of general compressors on log files. *ESE*, 25(5):3043–3085, 2020.

Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. Automated it system failure prediction: A deep learning approach. In *BigData 2016*, 2016.

Shenglin Zhang, Ying Liu, Dan Pei, Yu Chen, Xianping Qu, Shimin Tao, and Zhi Zang. Rapidand robust impact assessment of software changes in large internet-based services. In *ENET 2015*, 2015.

Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. Robust log-based anomaly detection on unstable log data. In *FSE 2019*, 2019.

A Dataset Details

In Table 3, we provide full details of datasets including the category of system, the number of messages, the number of anomaly log, the number of generated templates and the number of error types.

Table 3: 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.

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

B Comparison between Adapters

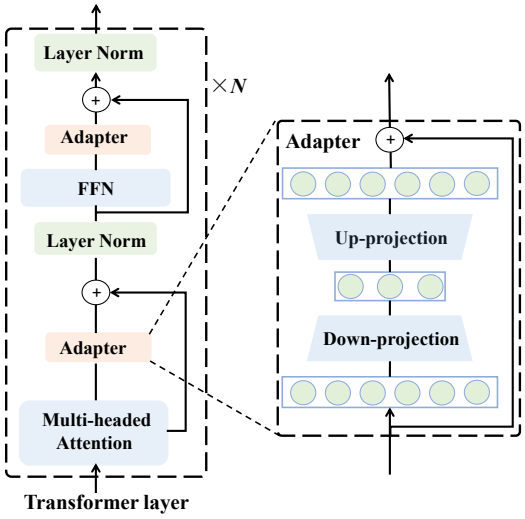

Figure 8: Original Serial Adapter. Where N is the number of transformer layers. The left part describes the transformer encoder inserted by adapters, the right part is the structure of an adapter.

In Fig. 8, the original adapter is inserted in Transformer in serial order, which is different from our log-adapter with parallel structure. Log-adapter has two strengths: 1) Log data is semi-structured data that retains partial semantic information, thus log adapter is designed to connect directly with the input, which aims to avoid losing the original semantics during the training. 2) The parallel structure is decoupled from the backbone model, preventing problems like gradient explosion, leading to a robust training procedure. We compare the F_1 scores of two types of adapters in Table 4. Results show that our log-adapter gains 1% improvement on three datasets on average, demonstrating the effectiveness of our log-adapter.

Table 4: F_1 scores between serial adapter and ours. Experiments are based on *4* layers of transformer encoder.

Adapter	HDFS	BGL	Thunderbird
Serial	0.988	0.984	0.983
Ours	**0.998**	**0.998**	**0.998**

C IMPLEMENTATION DETAILS

In the experiment, we try a different number 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 4 NVIDIA v100 with a batch size of 64. For each dataset, we tune the maximum learning of OneCycleLR scheduler in $\{1e-5, 5e-5, 1e-6\}$.

D TIME CONSUMPTION

Table 5 presents the training and testing time of TRANSLOG on HDFS, BGL, and Thunderbird, respectively. TRANSLOG gains the lowest time consumption compared with these baselines. TRANSLOG performs better than the state-of-the-art semi-supervised approaches (i.e., PLElog) and the state-of-the-art supervised approach (i.e., LogRobust). Since the training process is offline, the time consumption of TRANSLOG is acceptable. Despite the efficiency of all these methods, our method achieves the least testing time. In conclusion, TRANSLOG is efficient with short testing time and satisfactory offline training time.

Table 5: Time consumption of studied deep approaches. The lowest results are highlighted.

Method	HDFS		BGL		Thunderbird	
	Training	Testing	Trainging	Testing	Training	Testing
DeepLog	50m	10m	23m	6m	59m	12m
LogAnomaly	2h 50m	34m	2h 22m	25m	2h 43m	30m
PLElog	37m	35s	20m	14s	33m	31s
LogRobust	1h 20m	20m	40m	4m	58m	12m
TRANSLOG	**20m**	**20s**	**17m**	**11s**	**19m**	**16s**

E PRACTICAL EVALUATION

TRANSLOG has been successfully applied to a multinational cloud services company (the name is not shown due to the company policy). To test the generalization of TRANSLOG, we conduct the experiment on a real-world distributed online system called GAIA, which is considered to be released in the future. As this online system serves hundreds of corporations, 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), among them, there are 31,279 anomalous messages. In Table 6, TRANSLOG still performs best among baselines. Besides, TRANSLOG is stably running over 3000 hours on this system, which further demonstrates the stability of TRANSLOG.

Table 6: Experimental results compared with baselines on GAIA. The best results are highlighted.

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
	TRANSLOG	**0.89**	**0.98**	**0.93**