

Learning Representations on Logs for AIOps

Pranjal Gupta, Harshit Kumar, Debanjana Kar, Karan Bhukar, Pooja Aggarwal, Prateeti Mohapatra
IBM Research, India

Abstract—AI for IT Operations (AIOps) is a powerful platform that Site Reliability Engineers (SREs) use to automate and streamline operational workflows with minimal human intervention. Automated log analysis is a critical task in AIOps as it provides key insights for SREs to identify and address ongoing faults. Tasks such as log format detection, log classification, and log parsing are key components of automated log analysis. Most of these tasks require supervised learning; however, there are multiple challenges due to limited labeled log data and the diverse nature of log data. Large Language Models (LLMs) such as BERT and GPT3 are trained using self-supervision on a vast amount of unlabeled data. These models provide generalized representations that can be effectively used for various downstream tasks with limited labeled data. Motivated by the success of LLMs in specific domains like science and biology, this paper introduces a LLM for log data which is trained on public and proprietary log data. Results of our experiments demonstrate that the proposed LLM outperforms existing models on multiple downstream tasks. In summary, AIOps powered by LLMs offers an efficient and effective solution for automating log analysis tasks and enabling SREs to focus on higher-level tasks. Our proposed LLM, trained on public and proprietary log data, offers superior performance on multiple downstream tasks, making it a valuable addition to the AIOps platform.

Index Terms—AIOps, Log Analysis, Large Language Model

I. INTRODUCTION

With the growing prevalence of scalable microservices-based applications, log analysis is an integral part of building robust systems [1], [2]. As the scale of applications expands, the quantity of generated logs increases exponentially, posing a challenge for IT teams to analyze them manually [3]. One critical aspect of building resilient systems is log analysis. By analyzing logs generated by various components of the system, IT teams can detect issues, identify their root cause(s), and take corrective action before they impact system availability. In order to extract relevant information, log parsing is necessary, and log format detection plays a crucial role in this process. Accurately detecting the log format allows Site Reliability Engineers (SREs) to focus on the relevant logs and interpret log data effectively. After logs are parsed and collected, monitoring them is crucial for assessing the system’s health. Logs can be categorized into different “golden signals” [4] to facilitate this monitoring process. The combination of log format detection and golden signal classification can reduce the mean time to detect an issue. Additionally, accurate fault category prediction is crucial in reducing the mean time to engage the right expert to handle the fault. A deeper understanding and representation of logs plays a crucial role in providing key insights to SREs in detecting faults, improving system availability, and minimizing downtime.

One popular way of obtaining generalized representation for text is Large Language Models (LLMs). Recent years have seen the emergence of deep learning pre-trained LLMs such as BERT [5], GPT-3 [6], PaLM [7], DALL-E [8], and Stable Diffusion [9]. LLMs have been demonstrated to possess significant capabilities for representation learning and have the versatility to be utilized in diverse downstream applications, such as response generation, summarization, and text-to-image generation, etc. LLMs are built by infusing large amounts of data to train a deep neural network using one or more self-supervised auxiliary tasks. These models provide generalized representations that are not task-specific but can be fine-tuned for multiple downstream tasks, with limited labeled data in a few-shot setting. By leveraging the advanced representation learning capabilities of LLMs, log analysis can become even more efficient and effective, leading to significant improvements in application performance and reliability. Our focus is to utilize the log specific LLM on the three log analysis tasks: log format detection, golden signal classification, and fault category prediction.

Directly applying existing LLMs, which are pre-trained on natural language text, to semi-structured log data can be challenging due to the dissimilarities between the two data types. For instance, the lexicon and vocabulary of log data differs from natural language text; this is because log data contains domain-specific words and phrases that are not commonly used outside of their respective fields. As most log messages have a limited grammatical structure, the representations or embeddings provided by pre-trained LLMs do not provide the best results on downstream tasks. Also, capturing token positional information in logs is crucial as they help in log parsing tasks. Displayed in Table I are examples of log lines, their corresponding templated forms [10], and the labels assigned to them based on their downstream tasks. These examples demonstrate the crucial role of comprehending the inherent structure, vocabulary, and attributes of log messages in facilitating efficient log analysis. Similar observations have also been reported by several other prior works [11]–[14] that pre-trained LLMs do not perform as expected on domain-specific tasks.

Inspired by the LLMs for different domains, such as TweetBERT [15], SciBERT [16], and BioBERT [11], we introduce *BERTOps*, an LLM for AI in operations (AIOps) domain, pre-trained over large-scale public and proprietary log data. The emergence of domain-specific LLMs is due to the fact that existing LLMs are trained on general corpora such as news articles and Wikipedia, and hence do not perform as expected for specific domains. To the best of our knowledge,

TABLE I
SAMPLE LOG LINES WITH ANNOTATIONS FOR THE THREE DOWNSTREAM TASKS

LogLine	Template	Downstream Task Label		
		Log Format	Golden Signal	Fault Category
java.io.ioexception: Failed to create local dir in /opt/hdfs/nodemanager/usercache/curi/appcache/application	java.io.IOException: Failed to create local dir in <*>	Spark	Error	I/O
12-18 19:49:14.062 633 31868 e sdk: <2016-12-18 19:49:14>[ERR] SDK: UE-SeC 2016-12-18 19:49:14:62 Level[ERR] magic[3365677344 635]:TcpNonBlockReConnect failed because it can RW! err:110[Connection timed out]	12-18 <*>633 31868 E SDK : <*>[ERR] SDK: UE-SeC 2016-12-18 <*>Level[ERR] magic[3365677344 635]:TcpNonBlockReConnect failed because it can RW! err:110[Connection timed out]	Android	Latency	Network
48587 node-235 action error 1076691608 1 clusterAddMember (cmd 2162) Error: Cannot run clusterAddMember\because node-235 connection state is REFUSED, needs to be SRM	<*><*>action error <*>1 clusterAddMember (cmd <*>)Error: Cannot run clusterAddMember\because <*>connection state is REFUSED, needs to be SRM	HPC	Availability	Network

BERTOps is the first LLM that can effectively generalize to multiple downstream tasks of log analysis. We finetune the pre-trained *BERTOps* model under a few-shot setting for three downstream tasks - Log Format Detection (LFD), Golden Signal Classification (GSC), and Fault Category Prediction (FCP). For the aforementioned tasks, we show that *BERTOps* outperforms classical machine learning models and LLMs. The main contributions of this paper are as follows:

- 1) We propose an encoder-based LLM (*BERTOps*) for the AIOps domain, and show its application on the three downstream tasks for log analysis - Log Format Detection, Golden Signal Classification, and Fault Category Prediction.
- 2) We provide labeled data for the aforementioned three downstream tasks, and release it publicly along with the code base. We believe that these tasks along with the datasets and the model will work as a benchmark for further research. To the best of our knowledge, there are currently no existing labeled datasets available for the Golden Signal Classification task and Fault Category Prediction task.
- 3) Our experiments suggest that the encoder-based *BERTOps* exhibit few-shot generalization with purely unsupervised training. To demonstrate the effectiveness of LLM for AIOps, we compare the *BERTOps* with classical machine learning models and pre-trained encoder-based models such as BERT and RoBERTa. An important observation is a significant gain in performance of *BERTOps* vis-a-vis classical ML models for log analysis tasks.

II. METHOD

In this section, we introduce *BERTOps*, a large-scale Transformer model for log analysis and understanding. Figure 1 shows an end-to-end architecture of the *BERTOps*, including pretraining the model and finetuning it on the downstream tasks. The downstream tasks for log analysis are mostly classification-based, therefore we build an encoder-based LLMs for log data. The architecture design of *BERTOps* is motivated from *BERT-BASE* [5] with 12 layers, 12 self-attention heads, and the size of representation layer is 768.

Since there is a vocabulary overlap between natural text and log data, especially for non domain-specific words and phrases, the pretrained weights from BERT-BASE are used as the initialization weights for *BERTOps* and further pretrain it on logs data. The intuition is to bootstrap *BERTOps* with BERT’s knowledge of natural language text, so that it can focus more on understanding log data specific vocabulary and representation during pretraining. The transformer encoder of *BERTOps* is further pretrained on logs data using the masked language modeling (MLM) task [5]. During pretraining phase, some percentage of tokens in a log sequence are randomly masked as shown in Figure 1. The objective of the MLM task is to predict the token corresponding to the masked token based on the neighbouring context tokens. For example, Figure 1 demonstrates an instance of token masking in which the token “server” is masked. The objective of the MLM task is to utilize the embedding H_{masked} to predict the original token, “server”, by employing cross-entropy loss (L_{MLM}). We continue pretraining the *BERTOps* until training loss (L_{MLM}) saturates, and use the model with the least validation loss as the final model. For example, Figure 2 shows that at epoch 5, the validation loss is minimum. Also, note that the validation loss saturates between epochs 5 and 6. Although we continued training after epoch 6, hoping that loss would further reduce, however, that did not happen. Instead, it went up a little and in the following epoch, it came back to the same level which is a sign that the loss is converging. Therefore, we used the model checkpoint at epoch 6 as the final model for the downstream tasks. Besides using validation loss for model selection, we also used perplexity scores to monitor the progress during pretraining. Perplexity [17] measures how well a LLM predicts a sequence of words. The lower the perplexity score, the better the language model is at predicting the next word. It is calculated as follows:

$$PP(\mathbf{w}) = \sqrt[n]{\frac{1}{P(w_1, w_2, \dots, w_N)}}$$

where $\mathbf{w} = w_1, w_2, \dots, w_N$ is a sequence of words, and $P(w_1, w_2, \dots, w_N)$ is the joint probability of the sequence

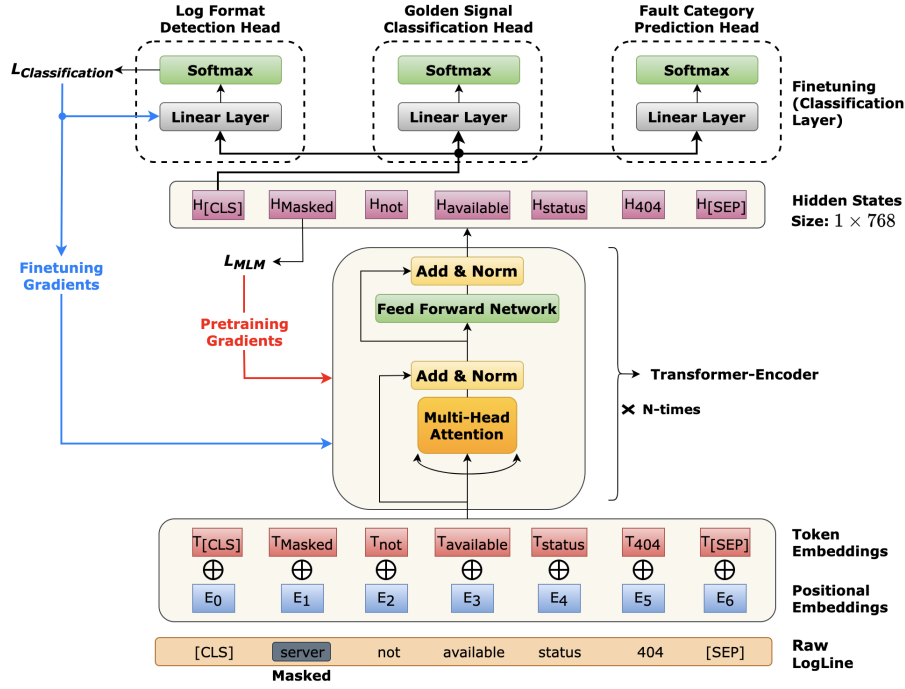


Fig. 1. BERTOps end-to-end architecture pre-training and finetuning on three downstream tasks. The model has been designed to be plug-and-play such that new downstream tasks can be added when required. It computes two losses: pretraining loss L_{MLM} and task specific finetuning loss $L_{classification}$.

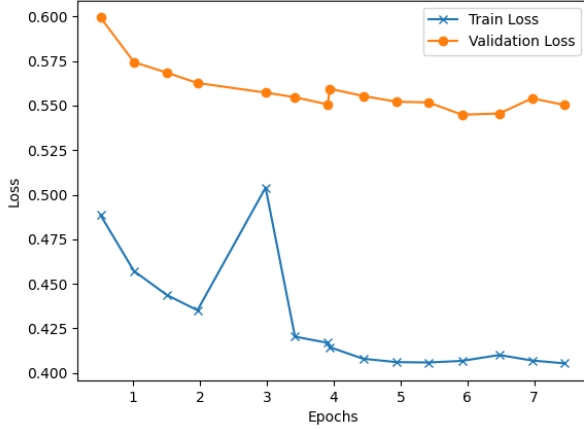


Fig. 2. Train vs Validation Loss during pretraining of BERTOps

of words to occur together. As the training continued, the perplexity score came down from 87.9937 at epoch 0 to 1.7363 at epoch 5.4, the same checkpoint which had the least validation loss. The high perplexity score in epoch 0 validates the hypothesis mentioned in the Introduction (Section I) that log lines do not have proper grammatical structure as opposed to natural language text. On further training, the perplexity score dropped, this suggests that the *BERTOps* adjusted its weights to build representations in accordance with the syntactical and semantical structure of words appearing in log data.

As mentioned earlier, pretraining LLMs requires a huge amount of data. For BERTOps, the pretraining data consists of logs from 17 different data sources, out of which 12

TABLE II
"PUBLIC" PRETRAINING DATASET STATISTICS

LogSource	Train	Validation	#Instances	#Templates
Android	1244K	311K	1555K	5,852
BGL	3798K	949K	4747K	584
HDFS	8940K	2235K	11175K	95
HPC	346K	86K	433K	208
Hadoop	315K	78K	394K	475
Linux	204K	5K	25K	196
Mac	93K	23K	117K	1188
Openstack	151K	37K	189K	158
Proxifier	17K	4K	21K	82
Spark	26589K	6647K	33236K	170
Thunderbird	80K	20K	100K	1465
Zookeeper	59K	14K	74K	61
Total	41.6M	10.4M	52.07M	10.5K

TABLE III
"PROPRIETARY" PRETRAINING DATASET STATISTICS

LogSource	Train	Validation	#Instances	#Templates
Apache	1719K	430K	2149K	372
HAProxy	4K	1K	5K	6
MongoDB	120K	30K	150K	14
Roboshop	17K	4K	21K	206
Sockshop	87K	22K	108K	179
Total	1.9M	487K	2435K	777

are collected from an open-source repository [18], and the remaining 5 are proprietary data sources. Tables II and III show the number of instances per log format for pretraining BERTOps. Proprietary data sources consist of log formats from custom applications whose structure are different from existing public log sources. The proprietary data sources

only constitute 0.937% of the entire pretraining corpus. Therefore, we can infer that using only public datasets for pretraining shouldn't have a significant deviation from the results presented in this paper. In order to generate the training and validation sets for our experiments, an 80 : 20 split was performed on each log source. Subsequently, the resulting data was combined to form the training and validation sets, which comprised of 43.5M and 10.9M log lines, respectively.

Data preprocessing of pretraining datasets consists of splitting tokens that are in camelcase format or tokens joined with periods or dash using regular expressions followed by converting tokens to lower case. The pretraining for BERTOps involved 8 epochs that spanned across a 20-day period. Huggingface's implementation¹ was used for pretraining BERTOps on 4 A100 GPUS with a batch size of 256. To obtain statistics for *Out-of-Vocabulary* (OOV) words in the pretraining dataset, we calculated the frequency of <unk> tokens (which signify OOV). Our analysis revealed that after utilizing BERT's tokenizer, 0.0062% and 0.0061% of tokens in the training and validation data, respectively, correspond to OOV. Once pretraining of BERTOps model is complete, it is finetuned using a cross-entropy-loss $L_{classification}$ for each of the task separately. More details for finetuning BERTOps are discussed in subsection III-C

III. EXPERIMENTAL SETUP

The usefulness of a pretrained LLM hinges upon how well it performs across multiple downstream tasks. That is, how effectively can it apply or transfer the knowledge it gained through pre-training on large corpuses for the downstream tasks. This section introduces three downstream tasks for log analysis in AIOps domain, process of preparing labeled datasets for these tasks, and finetuning the pretrained LLM on these tasks. The three tasks are: Log Format Detection, Golden Signal Classification, and Fault Category Prediction.

A. Downstream Tasks

To evaluate the effectiveness of the *BERTOps* model and the other baselines, three downstream log analysis tasks are defined.

- 1) **Log Format Detection (LFD)** Identifying the format of logs is an important first step of any log analysis pipeline [19]–[21], which can help leverage the knowledge of its unique structures to parse the logs and aid in tasks like structure extraction [22], key entity/metric extraction, anomaly detection [23], etc. With multiple log variations within each log format (refer column Templates in Table IV), learning to distinguish them from a few training samples makes it a challenging task. In this task, given logs from varied sources, we train a multi-class classification model that learns to distinguish logs from 16 different log formats. Tables II and III provides the number of templates (indicating variations) per format.

¹https://github.com/huggingface/transformers/blob/main/examples/pytorch/language-modeling/run_mlm.py

TABLE IV
SAMPLE LOG VARIATIONS IN SSH AND HDFS LOG FORMATS

Format	Template
HDFS	<*>INFO dfs.DataNode\$PacketResponder: PacketResponder 2 for block<*>terminating
	<*>INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock: <*><*>
	<*>INFO dfs.DataNode\$BlockReceiver: Exception in receiveBlock for block <*>java.io.IOException: Connection reset by peer
SSH	<*>LabSZ <*>Failed none for invalid user <*>from <*>port <*>ssh2
	<*>LabSZ <*>Invalid user <*>from <*>
	<*>LabSZ <*>Received disconnect from <*>11: Bye Bye [preauth]

- 2) **Golden Signal Classification (GSC)** Google SRE handbook for SREs [4] outlines basic principles and practices for monitoring applications hosted on cloud. Golden Signals are a set of key performance indicators (KPIs) used for monitoring log and metric events to analyze health of a system. One of the key benefits of monitoring the golden signals is that it offers a convenient and efficient method for detecting and troubleshooting errors. By monitoring these KPIs, it is possible to detect anomalies and trends that can indicate problems before they become critical. This enables teams to proactively respond to issues and ensure that their systems are running smoothly and efficiently. The four golden signals defined in the SRE handbook are latency, traffic, errors, and saturation. Latency measures the time it takes for a request to be processed, errors count the number of failed requests, saturation measures the degree to which system resources are being used, and traffic measure of how much demand is being placed on your system. We

TABLE V
LOG EXAMPLES WITH GOLDEN SIGNAL LABELS

LogLine	Golden Signal
java.net.NoRouteToHostException: No route to host: no further information	Availability
DateTime Got an XPC error: Connection invalid	Error
DateTime INFO Notification time out 60000	Latency
Caused by: org.apache.hadoop.fs.FSError: java.io.IOException: There is not enough space on the disk	Saturation
SohuNews.exe - _URL_ close, 673 bytes sent, 384 bytes received, lifetime 00:18	Traffic
12-18 15:33:40.463 3180 3578 i ash : application com.tencent.mm is im app and connected ok	Information

introduce "availability", a new golden signal to the set of golden signals. The intuition to introduce a new class is to capture a special type of error that needs immediate SRE attention. When a logline indicates an availability golden signal, it means that the underlying application is not responding or device is not reachable and may lead to a critical fault. Also, there are certain log messages that are non-erroneous, such as heart beat messages, trace messages, etc - such log messages do not belong to the

existing golden signals, we have defined an additional class for them called as "Information". Table V enumerates examples of each golden signal classification label.

- 3) **Fault Category Prediction (FCP)** Fault categories associated with logs serve as signals to detect anomalous system behaviours and provide clues for failure diagnosis. That is, a Fault category helps to understand why a service fails and may help in isolating the root causes of the failure. In addition, fault categories can also be used to route the ongoing fault/issue to appropriate teams for debugging and remediation. Through this task, we aim to build a fault categorization model that classifies a log line or raw log message into one of the 7 fault categories: Memory, Network, Authentication, I/O, Device, Application, and Other [24]. Table VI provides an example for each fault category.

TABLE VI
LOG EXAMPLES WITH FAULT CATEGORY LABELS

LogLine	Fault Category
WARN dfs.FSDataset: Unexpected error trying to delete block blk_6566051927569845875. BlockInfo not found in volumeMap.	Memory
Connection not available while invoking method createOrWaitForConnection for resource jdbc/Activ	Network
SSL HANDSHAKE FAILURE: A signer with SubjectDN was sent from a target host. The certificate expired at Mon Feb 15	Authentication
WARN [ResponseProcessor for block: Slow Read-Processor read fields took 48944ms (threshold=30000ms); SUCCESS status:	I/O
Server launched but failed initialization. Server logs, startServer.log, and other log files under.	Device
An exception was thrown by one of the service methods of the servlet in application	Application
081110 065758 7079 INFO dfs.DataNode\$DataXceiver: 10.251.106.10:50010 Served block blk_-6040371292695818218 to 10.251.74.134	Other

B. Labeled Datasets for Downstream Tasks

Training a model for a task from scratch requires huge amount of labeled data instances which are hard and expensive to generate. Since we aim to evaluate the effectiveness of the fine-tuned LLM for the downstream tasks with only a few labeled training examples, it is crucial to have an extensive labeled test set to validate its performance. This work aims to provide a labeled training dataset for fine-tuning models under a few-shot setting. Additionally, an annotated test dataset will be made available, which can serve as a suitable benchmark for these tasks, thereby enabling researchers to make comparisons and evaluate the performance of their models. Log data for the downstream tasks is sourced from LogHub [18] and proprietary data sources. While we had access to gold-standard annotations for the LFD task, we manually curated annotated data samples for the other two tasks.

For the LFD task, a dataset was prepared from the 16 formats consisting of Android, Apache, BGL, HDFS, HPC,

Hadoop, HealthApp, Mac, Openstack, Proxifier, SSH, Syslog-Sendmail, Spark, Thunderbird, Websphere and Zookeeper. Out of the 16 log formats, the 4 log formats (HealthApp, SSH, Sendmail, WebSphere) were treated as held-out, i.e., these 4 log formats were not used for pretraining *BERTOps* (refer Table II and III), but they will be required during finetuning the pretrained LLM models for the downstream tasks. The goal of holding out these 4 datasets is to test the generalizability of LLMs, including *BERTOps*, to handle unseen datasets for the downstream tasks.

Preparing labeled datasets for the GS Classification and the FC Predication tasks is a challenging process for two reasons: one, the amount of data is huge, and second, it is practically infeasible for human annotators to label each logline. To mitigate this situation, we templated logs for each format using the Drain template miner [22]. Templating allows to cluster logs into homogeneous groups that have the same structure. We distributed few instance of each log template among the annotators. For example, Table III shows that the number of log lines for MongoDB log format are approximately 150K which can be grouped into 14 templates. Instead of labeling each of the 150K log lines, the annotators were asked to label only 14 templates. The template labels were reverse mapped to actual log lines associated with each template. This reduced the human effort of manually labeling each log line by manifold while getting a good coverage.

For GS classification task, 7 subject matter experts were provided with 272 templates for labeling. The overlap of log templates among the annotators was 66%. Similarly, for FC prediction task, 394 templates were provided for labeling with an overlap of 43%. After labeling, it was observed that the Traffic class had a skewed distribution with very few examples. This skewed distribution ultimately diminished the overall quality of the dataset. As a result, the final labeled dataset does not include the Traffic class. Once the labeling of templates for GS and FC tasks was completed, those templates where annotators had disagreements were revisited to resolve the differences. During the disagreement resolution process, conflicts were resolved using majority vote. Kappa coefficient [25] is used as a metric for computing inter-annotator agreement. For the GSC and FCP, the inter-annotator agreement is 60.62 and 65.80, respectively. Inter-annotator agreement in the range of 60-65 indicates the complexity associated with these tasks.

To collate the k-shot training dataset for each task, where $k \in \{10, 20, 30\}$, we randomly selected k templates with discernible variations. We curate 10, 20 and 30 shot training datasets for each task by finding 10, 20, 30 samples of each label for each task. For example, 10-shot training set for log format detection will have 16×10 log samples. We used the rest of the templates along with their group of instances as the test set. For each of the three downstream tasks, the data statistics of the labeled dataset are presented in Table VII.

C. Finetuning

A widely used industry practice is to build an LLM which upon release is adopted and deployed for the downstream

TABLE VII
DOWNSTREAM TASKS DATA DISTRIBUTION

Task	# Classes	# Test
Log Format Detection	16	11918
Golden Signal Classification	5	423
Fault Category Prediction	7	868

tasks by application-focused developers and businesses. The pre-trained *BERTOps* model prepared in Section II is further finetuned for each task separately. Finetuning a model for a particular task requires labelled data. In the real-world scenario, clients find it difficult to share large data payloads with private data to train models for various tasks. The other challenge is the availability of labeled data. Moreover, data annotation for each task is an expensive and laborious exercise. Few-shot learning mitigates that bottleneck by teaching the models to classify with very few data samples.

For the downstream tasks described in Section III-A, we finetune *BERTOps* in a few-shot setting to evaluate the effectiveness of our learned transformer representations. As shown in Figure 1 we add a classification layer on top of *BERTOps* consisting of a linear layer followed by softmax activation. We use *simpletransformers*² library for performing experiments on downstream tasks. All the downstream tasks use Cross Entropy loss as the objective function. The finetuning training process updates all model parameters including the pretrained weights. These experiments are performed on a single A100 GPU server for 20 epochs with AdamW optimizer [26] and learning rate of $4e-5$.

IV. EXPERIMENTS

In this section, we present results of our experimental study, followed by its analysis. The experimental studies are designed to answer the following questions: (1) Is an AIOps domain specific LLM required? (2) How well does the proposed *BERTOps* LLM performs in comparison to the other baseline models for the three downstream tasks, i.e. Log Format Detection, Golden Signal Classification, and Fault Category Prediction tasks? (3) How effective is the few-shot learning approach?

A. Baseline Models

In this section, we list the baseline models that we use for comparison in our experiments. LLMs such as BERT are known to outperform LSTMs or RNNs [27], [28], therefore we do not include them in our study. To answer the first question, we compare the performance of *BERTOps* with state-of-the-art LLMs such as ALBERT-base-v2 [29], ELECTRA [30], XLNET [31], RoBERTa [32], and BERT-base [5]. For most AIOps tasks, the state-of-the-art models are either rule-based or ML-based [33], [34], and to answer the second question, we include two ML models as baselines, Decision Tree(DT) and Stochastic Gradient Descent (SGD). These models were

trained using scikit-learn library³ for each of the three downstream tasks. To answer the third question, we run experiments under the few-shot setting with $k \in \{10, 20, 30\}$ (refer Section III-B and III-C for details on finetuning).

B. Results

Tables VIII - X presents weighted Precision, Recall and F1-scores obtained on 10, 20 and 30 shot datasets used for finetuning *BERTOps* on the three downstream tasks. We observe that *BERTOps* outperforms all pretrained LLMs and ML models on the three datasets, except on the Fault Category Prediction when only 10 examples are provided for finetuning.

For the LFD task, the proposed *BERTOps* LLM learns faster with minimal training data as opposed to other models. Even with as few as 10 samples per format provided for finetuning, *BERTOps* achieves an F1 score of 97.23%. Whereas, when 30 examples were provided for finetuning, the best ML model (SGD) and the best LLM model(RoBERTa) has a weighted F1-score of 90.34% and 98.3% which is very close to *BERTOps* trained with 10 examples. With 30 shot dataset, *BERTOps* shows percentage improvement of 9.98% and 1.08% when compared with the best performing ML (SGD) and pre-trained LLM(RoBERTa), respectively.

In the GS Classification task, we observe significant improvements in F1 score with LLMs vis-a-vis the ML models. When the number of training samples per class increase from 20 to 30, the best performing ML model (SGD) improves by approximately 4.1 absolute points (F1 from 67.55 to 71.65). Whereas, the best performing LLM model(RoBERTa) and *BERTOps* has an improvement of 7.92 and 8.81 absolute points, respectively. These results indicate that *BERTOps* being a domain specific LLM not only outperforms existing state-of-the-art models, but it also gets a good start (58.95 vs 50.91 for SGD and 58.95 vs 56.03 for the BERT-base). The percentage improvements of *BERTOps* with respect to the best performing ML(SGD) and LLM (RoBERTa) on 30 shot dataset are 9.28% and 1.65%, respectively.

In the FC Prediction task, *BERTOps* doesn't record a performance gain when finetuned on smaller dataset size of 10 examples, however, it outperforms all baseline methods when finetuned with a larger dataset size of 20 or 30 examples. Also, when finetuning with 10 examples, although ALBERT-base has the highest F1-score, *BERTOps* outperforms all baseline models in terms of precision. This means *BERTOps* exhibited higher precision in identifying the correct fault for a logline, which is an important metric in the FC Prediction task. Precision measures the ratio of number of correct predictions to the total number of predictions. A higher precision means that the *BERTOps* has less false positives as compared to other methods. On the other hand, Recall measures the ratio of total number of correct predictions to the actual ground truth. A lower recall and a higher precision indicates that *BERTOps* failed to identify some of the fault category classes, but among the ones that it identified, those were actually correct. Also,

²<https://simpletransformers.ai/>

³<https://scikit-learn.org/stable/>

TABLE VIII
COMPARING BASELINES WITH BERTOPS ON LOG FORMAT DETECTION

Log Format Detection										
Model Type	Model	10-shot			20-shot			30-shot		
		P	R	F1	P	R	F1	P	R	F1
Classical ML	Decision Tree	93.49	71.86	75.10	93.05	77.57	81.35	92.08	83.34	85.08
	SGD	89.10	87.15	87.17	90.21	87.98	88.04	91.87	90.35	90.34
Pretrained LLMs	ALBERT-base-v2	95.54	94.26	94.58	96.79	96.27	96.40	98.20	98.19	98.19
	ELECTRA	89.53	85.87	86.24	95.67	94.74	94.94	98.03	97.97	97.98
	XLNet	95.34	94.76	94.83	96.41	96.24	96.27	97.96	97.77	97.81
	RoBERTa	95.17	94.24	94.42	96.72	96.49	96.55	98.31	98.30	98.30
	BERT-base	95.61	95.17	95.23	96.96	96.81	96.84	97.63	97.42	97.47
Proposed Domain Specific LLM	BERTops	97.37	97.21	97.23	97.97	97.90	97.91	99.38	99.35	99.36

TABLE IX
COMPARING BASELINES WITH BERTOPS ON GOLDEN SIGNAL CLASSIFICATION

Golden Signal Classification										
Model Type	Model	10-shot			20-shot			30-shot		
		P	R	F1	P	R	F1	P	R	F1
Classical ML	Decision Tree	47.82	35.46	36.55	64.86	63.36	63.14	54.40	52.72	52.37
	SGD	56.17	50.35	50.91	70.75	66.90	67.55	72.74	71.16	71.65
Pretrained LLMs	ALBERT-base-v2	54.39	49.88	50.02	66.21	62.41	62.97	78.23	73.52	74.81
	ELECTRA	50.79	47.75	48.44	66.30	59.34	59.96	75.27	74.70	74.69
	XLNet	55.96	51.54	50.38	69.71	66.43	66.60	76.25	75.18	75.42
	RoBERTa	60.64	56.03	55.75	72.93	68.32	69.11	78.66	76.60	77.03
	BERT-base	61.37	56.50	56.03	73.54	66.19	67.51	77.34	75.89	76.20
Proposed Domain Specific LLM	BERTops	62.15	58.39	58.95	70.90	69.03	69.49	78.51	78.25	78.30

TABLE X
COMPARING BASELINES WITH BERTOPS ON FAULT CATEGORY PREDICTION

Fault Category Prediction										
Model Type	Model	10-shot			20-shot			30-shot		
		P	R	F1	P	R	F1	P	R	F1
Classical ML	Decision Tree	72.38	46.54	50.55	74.41	47.12	48.49	72.87	52.88	55.38
	SGD	72.04	49.19	52.07	75.85	59.951	62.31	78.31	62.67	64.94
Pretrained LLMs	ALBERT-base-v2	76.01	62.90	65.85	77.19	60.83	63.73	78.13	68.32	69.83
	ELECTRA	75.17	57.95	62.03	78.54	65.32	68.15	79.64	67.28	69.63
	XLNet	75.24	53.57	56.82	76.46	57.83	59.83	80.67	70.05	72.06
	RoBERTa	76.71	51.73	53.38	80.35	69.70	71.72	80.49	66.82	69.05
	BERT-base	76.22	55.18	57.94	78.68	59.56	61.69	80.79	72.35	74.00
Proposed Domain Specific LLM	BERTops	77.33	50.81	54.70	79.72	70.16	71.88	81.84	74.65	76.12

it is impressive to observe that BERTops quickly learned and corrected itself when provided with more training examples. That is, when 30 training examples were provided for finetuning, *BERTops* achieves an absolute increase of 23.84 points in recall, which is the highest improvement observed compared to all baseline models. Whereas, ALBERT-base improves only by 5.42 absolute points. We observe a similar trend in F1-score, where BERTops and ALBERT-base improves by 21.42 and 3.98 absolute points, respectively.

Also worth noting is the inter-annotator Kappa score and the F1 score of the ML models for the GS and FC classification tasks. The inter-annotator agreement was in the range of 60-65, and the F1 score of ML models for both tasks is in the range of 50-70. Generally, it is believed that inter-annotator agreement places an upper bound on the performance of the ML models [35], which is also evident in these results. Our experiments also indicate an intriguing result where the classical ML models outperforms the state-of-the-art LLM baselines. Specifically, for the GS Classification task with

20-shot dataset for finetuning, the SGD outperforms most of the LLM models such as ALBERT-base, ELECTRA, XLNet, and BERT-base. A similar trend is observed for the FC Prediction task with 20-shot dataset for finetuning, where SGD outperforms both XLNet and BERT. One possible reason for this is the challenges associated with the log data which we also highlighted in the introduction section. Note that, BERT and other LLMs are pre-trained on natural language text, and hence they fail to correctly work with log data with few examples for finetuning on downstream tasks. Because *BERTops* is pre-trained on log data, it has an obvious advantage in working with log data for the downstream tasks. Moreover, when more training examples are provided to *BERTops*, it learns quickly and shows significant jumps in performance as opposed to other LLMs. Higher performance of *BERTops* as compared to both ML and existing LLM models and its ability to learn quickly when more training examples are provided indicate that a LLM for logs is warranted. The experimental findings suggest that *BERTops* is capable of providing a more

accurate generalizable representation for a logline, which can be effectively utilized for various downstream tasks in AIOps.

V. DISCUSSION

This section presents a detailed analysis of *BERTOps* for each downstream task at the individual class level. The analysis includes a qualitative evaluation of *BERTOps*, finetuned using a 30-shot dataset. In addition, we provide examples of predictions generated by *BERTOps* for the three tasks, juxtaposed with BERT’s predictions.

For the LFD task, Table XI shows a truncated confusion matrix; to save space it is showing only 6 out of 16 classes where predictions are not 100% accurate, i.e., for the other 10 classes not shown in this table, the predictions were absolutely correct. Among the four held-out log formats (Section III-B) - the performance on HealthApp, SSH and SendMail is 100%, while on WebSphere(WS) the performance is 99.18%. One reason for this superior performance could be the pretraining of *BERTOps* on varying log lines from 17 log formats that enabled it to learn robust and generalized representations. The LFD model is most confused between the following classes: Mac, Spark, and WebSphere. To understand this behavior, Table XIV analyzes 7 log lines with ground truth label, along with the predictions of *BERTOps* and BERT-base.

TABLE XI

CONFUSION MATRIX FOR THE LOG FORMAT DETECTION TASK USING *BERTOps*; ROW AND COLUMN CORRESPOND TO THE TRUE AND PREDICTED LABEL, RESPECTIVELY

	Android	Hadoop	HealthApp	Mac	Spark	WebSphere
Android	99.84	0.0	0.11	0.0	0.05	0.0
Hadoop	0.0	99.06	0.0	0.44	0.15	0.36
HealthApp	0.0	0.0	100.0	0.0	0.0	0.0
Mac	0.0	0.0	0.0	98.45	0.36	1.19
Spark	0.0	1.15	0.0	0.57	97.13	1.15
WebSphere	0.34	0.12	0.34	0.02	0.0	99.18

TABLE XII

CONFUSION MATRIX FOR THE GOLDEN SIGNAL CLASSIFICATION USING *BERTOps*; ROW AND COLUMN CORRESPOND TO THE TRUE AND PREDICTED LABEL, RESPECTIVELY

	Availability	Error	Information	Latency	Saturation
Availability	67.61	21.13	9.86	1.41	0.0
Error	11.11	77.12	8.5	3.27	0.0
Information	5.26	6.02	85.71	1.5	1.5
Latency	6.06	12.12	3.03	78.79	0.0
Saturation	0.0	18.18	6.06	0.0	75.76

While BERT-base misclassified log lines 1-3, *BERTOps* identified subtle nuances in log data to make correct predictions, which is critical for accurately identifying log formats. For example, the term *websphere* appears twice in the example log line number 3, BERT-base failed to recognize it and misclassified it as of type Spark. Whereas, *BERTOps* predicted the correct label *WebSphere* for it. Note that the term *websphere* is not appearing as an independent term in the log line, it is part

TABLE XIII

CONFUSION MATRIX FOR THE FAULT CATEGORY PREDICTION TASK USING *BERTOps*; ROW AND COLUMN CORRESPOND TO TRUE AND PREDICTED LABEL, RESPECTIVELY. ABBREVIATED CLASSES ARE APPLICATION(APP), AUTHENTICATION(AUTH)

	App	Auth	Device	I/O	Memory	Network	Other
App	76.24	9.9	0.0	0.99	0.0	5.94	6.93
Auth	0.0	82.61	0.0	0.0	0.0	4.35	13.04
Device	9.3	2.33	74.42	0.0	6.98	4.65	2.33
I/O	0.0	0.0	0.0	100.0	0.0	0.0	0.0
Memory	0.0	0.0	0.0	0.0	100.0	0.0	0.0
Network	8.54	2.44	0.0	6.1	0.0	73.17	9.76
Other	10.04	1.94	2.29	0.18	4.93	11.09	69.54

of a phrase *com.ibm.websphere.security*. Existing pretrained LLMs such as BERT fail to interpret such tokens resulting in wrong predictions. We mentioned some of the challenges associated with log data in the Introduction Section I, one such challenge is the lexicon and grammatical structure of log lines. To address these challenges, we preprocessed log data for pretraining the *BERTOps* model(Section III-B), i.e., using regular expressions to split tokens that are in camelcase format or tokens joined by periods. The preprocessing has helped *BERTOps* to learn better representation for log lines which is the reason why it identified the log format correctly in these examples. These examples demonstrate the ability of *BERTOps* to capture contextual information in log data, which is crucial for precise log format detection.

Among the five classes in the GS Classification task, *BERTOps* had the least confusion in identifying the Information class (Table XII). To enhance incident resolution in AIOps, it is crucial to minimize false positives in the informational class as they can impede the fault diagnosis process (Section III-A). This result indicates that *BERTOps* is able to reduce false alarm rate which is very useful for an SRE.

However, *BERTOps* performed relatively poorly in correctly identifying log lines of type Availability, 21.13% of them are misclassified as Error class. The confusion matrix also shows that *BERTOps* is confusing 12.12% Latency examples and 18.18% of Saturation examples with the Error class. To understand the reason for this behavior, we identified some of the examples listed in Table XV. It shows five log lines, its true labels and predictions from *BERTOps* and BERT. The first three examples demonstrate that *BERTOps* correctly identified golden signals by focusing on domain-specific keywords in the log lines, such as "time out" (first example), "unavailable state" (second example) and "action error" or "Error: Unknown" (third example). However, in the fourth example, *BERTOps* might have paid more attention to the phrase "Unexpected error" compared to the phrase "BlockInfo not found", leading to an incorrect prediction class Error whereas the actual class is Availability. Similarly, in the fifth example, *BERTOps* might have given more weight to phrase "service() exception", which resulted in misclassification. The predicted class is Error whereas the actual class is Saturation. Ideally, it should have focused on "java.lang.OutOfMemoryError",

TABLE XIV
QUALITATIVE ANALYSIS OF BERTOPS FOR LOG FORMAT DETECTION

Log Line	True Labels	BERTOps	BERT-base
1) Caused by: java.nio.channels.UnresolvedAddressException	Hadoop	Hadoop	Websphere
2) WebContainer: 9 ERRORPLUS br.com.plusoft.webapp.PlusoftFilter - javax.servlet.ServletException: org.apache.commons.fileupload.FileUploadBase\$IOFileUploadException: Processing of multipart/form-data request failed. Async operation timed out	Websphere	Websphere	Hadoop
3) EST 2/18/20 9:45:46:006 EST com.ibm.websphere.security.PasswordCheckFailedException com.ibm.websphere.security.Cust	Websphere	Websphere	Spark
4) RetryExec I org.apache.http.impl.execchain.RetryExec execute I/O exception (java.net.SocketException) caught when processing request to {s} -> https://ilesb.southindia.cloudapp.azure.com:443: Connection reset	Websphere	Hadoop	Hadoop
5) java.io.IOException: Failed to create local dir in /opt/hdfs/nodemanager/usercache/curi/appcache/application_1448006111297_0137/blockmgr-bcb1cfe2-beb8-4f46-b500-2a6945695a47/04	Spark	Hadoop	Hadoop
6) org.apache.avro.AvroTypeException: Attempt to process a enum when a union was expected	Hadoop	Websphere	Hadoop
7) java.io.FileNotFoundException: /opt/hdfs/nodemanager/usercache/curi/appcache/application_14480_0137/blockmgr-518b4be8-5335-426e-8669-d85c295c8087/29/shuffle_0_10_0.index.4ec675d8-4116-4cef-8c74-9a088a849e36 (No such file or directory)	Spark	Hadoop	Hadoop

TABLE XV
QUALITATIVE ANALYSIS OF BERTOPS FOR GOLDEN SIGNAL CLASSIFICATION

Log Line	True Labels	BERTOps	BERT-base
1) 2015-07-29 19:53:10,510 - INFO [QuorumPeer[myid=3]/0:0:0:0:0:0:2181:FastLeaderElection@774] - Notification time out: 800	Latency	Latency	Latency
2) 465980 node-129 unix.hw state_change.unavailable 1145652795 1 Component State Change: Component is in the unavailable state (HWID=4709)	Availability	Availability	Error
3) 19272 node-100 action error 1100785272 1 boot (command 3314) Error: Unknown We failed to determine the state of this node	Error	Error	Availability
4) 081109 213838 19 WARN dfs.FSDataset: Unexpected error trying to delete block blk_692167975271429163. BlockInfo not found in volumeMap.	Availability	Error	Error
5) ServletWrapper E com.ibm.ws.webcontainer.servlet.ServletWrapper service Uncaught service() exception thrown by servlet FinGetMessageServlet: java.lang.OutOfMemoryError: PermGen space	Saturation	Error	Error
6) J2CA1010E : Connection not available; timed out waiting for 180 seconds.	Availability	Latency	Latency

TABLE XVI
QUALITATIVE ANALYSIS OF BERTOPS FOR FAULT CATEGORY PREDICTION

Log Line	True Labels	BERTOps	BERT-base
1) WWLM0061W : An error was encountered sending a request to cluster member CELL-NAME=wasp2cell, MEMBERNAME=wasp2b, NODENAME=wasp23anodep2 and that member has been marked unusable for future requests to the cluster	Network	Network	Application
2) Jan 4 13:42:21 LabSZ sshd[32136]: error: Received disconnect from 212.83.176.1: 3: org.vngx.jsch.userauth.AuthCancelException: User authentication canceled by user [preauth]	Authentication	Authentication	Network
3) - 1118787050 2005.06.14 R16-M1-N2-C:J04-U11 2005-06-14-15.10.50.992809 R16-M1-N2-C:J04-U11 RAS KERNEL INFO 1 torus processor sram reception error(s) (dcr 0x02fc) detected and corrected	Device	Memory	Memory
4) 12-17 21:40:04.059 633 692 I TCP : tcp.conn4485 TcpClientThread.onRecv(TcpClientThread.java:522) tcp connection event TCP_CONNECTED	Other	Network	Other

which is a more indicative signal of the saturation signal. The sixth example is particularly interesting because it contains phrases related to both availability (*Connect not available*) and latency (*timed out waiting*) classes, making it difficult for the model to assign a single golden signal. While the annotators chose availability as the correct label, *BERTOps* predicted it as Latency, which highlights the ambiguity and complexity of this task. This example underscores the difficulty of precisely assigning golden signal labels to log lines that possess subtle nuances and multiple signals, to the point of even confounding human experts.

For the FC Prediction task (TableXIII), we found that *BERTOps* was highly accurate in identifying log lines related to memory and I/O faults, and the second-best prediction

is observed in authentication (Auth) class. *BERTOps* also demonstrated similar accuracy in identifying application(App), device, and network faults. However, it was confused between the following pairs of class labels, Auth with Other(13.04%), Network with App(8.54%) and Other(9.76%), and Other with App(10.04%) and Network(11.09%); it appears that the model was confused between the App, Other and Network classes. Table XVI shows examples of some of these cases with true labels along with predictions from *BERTOps* and BERT-base. In the first two examples, *BERTOps* correctly identified the fault category by leveraging domain-specific phrases, such as associating phrase "*sending a request*" with class Network in the first example, and associating phrase "*User authentication canceled*" with class Auth in the second example. However,

in the third example, BERTOps may have interpreted *SRAM* as a type of memory, leading to a prediction of memory class, whereas the annotators labeled it as device class. This example highlights the complexity of this task, which can be difficult even for a human to classify it.

The last example related to FC prediction is worth noticing, *BERTOps* excels in identifying fault category in log lines that exhibit abnormal behavior⁴ in log lines, but struggles to classify fault categories when there is no such behavior present. For example, the last example in Table XVI shows that *BERTOps* identified the context (Network), but failed to identify the correct class label Other, this is because the log line is not related to a failure. Based on the above analysis, it can be inferred that while BERTOps performs commendably, and it even outperforms state-of-the-art baselines, there is still scope for improvement in handling the intricacies of AIOps tasks.

VI. RELATED WORK

Existing LLMs are based on transformer architecture [36], and they can be broadly categorized into three types: encoder-based, decoder-based, and encoder-decoder-based. Encoder-based LLMs consist of an embedding layer followed by a stack of encoder layers, i.e., they use only the encoder layer of the transformer architecture. Examples of encoder-based LLMs are BERT [5], RoBERTa [32], Electra [30], XLNet [31], etc. Decoder-based LLMs consist of an embedding layer followed by a stack of decoder layers, for example, GPT3 [6], PaLM [7], BLOOM [37], etc. These models are autoregressive at each step, i.e. the input to the model is the output of the previous step, which it uses to generate the output for the following step. Encoder-Decoder-based LLMs consist of both the encoder layer and the decoder layer. Examples of encoder-decoder based LLMs are BART [38], T5 [39], etc. Encoder-based LLMs are suited for Natural Language Understanding (NLU) tasks, such as classification. The decoder-based and encoder-decoder-based LLMs are suited for generation tasks such as summarization, translation, etc. Our proposed LLM Model for logs is an encoder-based model, this is because the majority of the downstream tasks only need representational information for classifying a log line into a relevant class.

Rule-based approaches are commonly applied for the three downstream tasks Log Format Detection, Golden Signal Classification, and Fault Category Detection. Log aggregators like LogDNA [40] and Splunk [41] use manually curated rules for log format detection and log parsing. Nagar et. al. [42] used a dictionary-based approach that built rules using automatically created dictionaries for each golden signal category. The dictionaries were built using an in-house IT domain-specific Glove word embedding model. Zou et. al. [24] proposed a fault category detection system using manually defined regular expressions and custom built dictionary. However, writing a set of regular expressions can be expensive and difficult to

scale and maintain, requiring significant manual effort and engineering. The process of curating new rules for each new log format or previously unseen log can be arduous and time-consuming.

Recently, research works on using transformers for log analysis have emerged. LogBERT, BERT-Log, NeuralLog are BERT-based models trained specifically for the task of log anomaly detection [43]–[45]. These models are not truly large language models as they are pre-trained specifically for log anomaly detection task and are not applied to any other downstream tasks. We are the first to propose a LLM for logs, utilizing a transformer architecture and fine-tuning on multiple downstream tasks.

VII. CONCLUSION AND FUTURE WORK

This paper proposes a Transformer-based Large Language Model for AIOps (*BERTOps*) which is trained on a large corpus of public and proprietary log data. We finetuned the pre-trained *BERTOps* model on three downstream log analysis tasks: Log Format Detection, Golden Signal Classification, and Fault Category Prediction. Our experiments show that the proposed *BERTOps* LLM is competitive to both traditional ML and state-of-the-art generic LLM models on the three downstream tasks, that too with minimum training data. And, we observe significant improvements as more training data is provided. For future work, we will apply *BERTOps* on other AIOps tasks such as log parsing, log anomaly detection, incident prediction, and incident prioritization; this will also require defining additional auxiliary tasks for pretraining. In the AIOps domain, there are other modalities of data such as metrics and request traces. We also plan to add data from different modalities during training to produce a holistic and robust *BERTOps* for all AIOps downstream tasks.

REFERENCES

- [1] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surv.*, vol. 54, no. 6, jul 2021. [Online]. Available: <https://doi.org/10.1145/3460345>
- [2] D. Das, M. Schiewe, E. Brighton, M. Fuller, T. Černý, M. Bures, K. Frajták, D. Shin, and P. Tisnovsky, "Failure prediction by utilizing log analysis: A systematic mapping study," 10 2020, pp. 188–195.
- [3] R. Mahindru, H. Kumar, and S. Bansal, "Log anomaly to resolution: Ai based proactive incident remediation," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 1353–1357.
- [4] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*, 2016. [Online]. Available: <http://landing.google.com/sre/book.html>
- [5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: <https://aclanthology.org/N19-1423>
- [6] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: <https://arxiv.org/abs/2005.14165>

⁴Abnormal behavior may include errors, warnings, exceptions, and other unexpected behaviors that may indicate a problem or fault in the system

- [7] H. Peng, R. Schwartz, and N. A. Smith, "PaLM: A hybrid parser and language model," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3644–3651. [Online]. Available: <https://aclanthology.org/D19-1376>
- [8] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," 2021. [Online]. Available: <https://arxiv.org/abs/2102.12092>
- [9] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," 2021.
- [10] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '19. IEEE Press, 2019, p. 121–130. [Online]. Available: <https://doi.org/10.1109/ICSE-SEIP.2019.00021>
- [11] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "BioBERT: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 09 2019. [Online]. Available: <https://doi.org/10.1093/bioinformatics/btz682>
- [12] I. Chalkidis, M. Fergadiotis, P. Malakasiotis, N. Aletras, and I. Androutsopoulos, "Legal-bert: The muppets straight out of law school," *arXiv preprint arXiv:2010.02559*, 2020.
- [13] E. Alsentzer, J. R. Murphy, W. Boag, W.-H. Weng, D. Jin, T. Naumann, and M. McDermott, "Publicly available clinical bert embeddings," *arXiv preprint arXiv:1904.03323*, 2019.
- [14] R. Zhang, W. Yang, L. Lin, Z. Tu, Y. Xie, Z. Fu, Y. Xie, L. Tan, K. Xiong, and J. Lin, "Rapid adaptation of bert for information extraction on domain-specific business documents," *arXiv preprint arXiv:2002.01861*, 2020.
- [15] D. Q. Nguyen, T. Vu, and A. Tuan Nguyen, "BERTweet: A pre-trained language model for English tweets," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 9–14. [Online]. Available: <https://aclanthology.org/2020.emnlp-demos.2>
- [16] I. Beltagy, K. Lo, and A. Cohan, "SciBERT: A pretrained language model for scientific text," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3615–3620. [Online]. Available: <https://aclanthology.org/D19-1371>
- [17] L. R. Bahl, F. Jelinek, and R. L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, no. 2, pp. 179–190, 1983.
- [18] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics," *arXiv:2008.06448 [cs]*, Aug. 2020.
- [19] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, "Abstracting execution logs to execution events for enterprise applications (short paper)," in *2008 The Eighth International Conference on Quality Software*, 2008, pp. 181–186.
- [20] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "A lightweight algorithm for message type extraction in system application logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 11, pp. 1921–1936, 2012.
- [21] S. Messaoudi, A. Panichella, D. Bianculli, L. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 2018, pp. 167–16710.
- [22] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," in *2017 IEEE International Conference on Web Services (ICWS)*, 2017.
- [23] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1448–1460.
- [24] D. Zou, H. Qin, H. Jin, W. Qiang, Z. Han, and X. Chen, "Improving log-based fault diagnosis by log classification," in *Network and Parallel Computing*, C.-H. Hsu, X. Shi, and V. Salapura, Eds., 2014.
- [25] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960. [Online]. Available: <https://doi.org/10.1177/001316446002000104>
- [26] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017. [Online]. Available: <https://arxiv.org/abs/1711.05101>
- [27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [28] C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He *et al.*, "A comprehensive survey on pretrained foundation models: A history from bert to chatgpt," *arXiv preprint arXiv:2302.09419*, 2023.
- [29] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.
- [30] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.
- [31] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. Red Hook, NY, USA: Curran Associates Inc., 2019.
- [32] L. Zhuang, L. Wayne, S. Ya, and Z. Jun, "A robustly optimized BERT pre-training approach with post-training," in *Proceedings of the 20th Chinese National Conference on Computational Linguistics*. Huhhot, China: Chinese Information Processing Society of China, Aug. 2021, pp. 1218–1227. [Online]. Available: <https://aclanthology.org/2021.ccl-1.108>
- [33] S. Nagar, S. Samanta, P. Mohapatra, and D. Kar, "Building golden signal based signatures for log anomaly detection," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, 2022, pp. 203–208.
- [34] D. Zou, H. Qin, H. Jin, W. Qiang, Z. Han, and X. Chen, "Improving log-based fault diagnosis by log classification," in *Network and Parallel Computing: 11th IFIP WG 10.3 International Conference, NPC 2014, Ilan, Taiwan, September 18-20, 2014. Proceedings 11*. Springer, 2014, pp. 446–458.
- [35] M. Boguslav and K. B. Cohen, "Inter-annotator agreement and the upper limit on machine performance: Evidence from biomedical natural language processing," *Studies in health technology and informatics*, vol. 245, pp. 298–302, 2017.
- [36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.
- [37] BigScience Workshop, "BLOOM (revision 4ab0472)," 2022. [Online]. Available: <https://huggingface.co/bigscience/bloom>
- [38] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 7871–7880. [Online]. Available: <https://aclanthology.org/2020.acl-main.703>
- [39] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 1, jan 2020.
- [40] "Automatically parsed logline components," <https://docs.mezmo.com/docs/log-parsing>.
- [41] "Splunk," <https://docs.splunk.com/>.
- [42] S. Nagar, S. Samanta, P. Mohapatra, and D. Kar, "Building golden signal based signatures for log anomaly detection," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, 2022, pp. 203–208.
- [43] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–8.
- [44] S. Chen and H. Liao, "Bert-log: Anomaly detection for system logs based on pre-trained language model," *Applied Artificial Intelligence*, vol. 36, no. 1, p. 2145642, 2022. [Online]. Available: <https://doi.org/10.1080/08839514.2022.2145642>
- [45] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 492–504.