End-to-End AutoML for Unsupervised Log Anomaly Detection

Shenglin Zhang
Nankai University & HL-IT
Tianjin, China
zhangsl@nankai.edu.cn

Yuhe Ji
Nankai University
Tianjin, China
jiyuhemail@foxmail.com

Jiaqi Luan
Nankai University
Tianjin, China
2120230752@mail.nankai.edu.cn

Xiaohui Nie
CNIC, CAS
Beijing, China
xhnie@cnic.cn

Zi'ang Chen
Nankai University
Tianjin, China
2012217@mail.nankai.edu.cn

Minghua Ma
Microsoft
Redmond, WA, USA
minghuama@microsoft.com

Yongqian Sun*
Nankai University & TKL-SEHCI
Tianjin, China
sunyongqian@nankai.edu.cn

Dan Pei
Tsinghua University
Beijing, China
peidan@tsinghua.edu.cn

ABSTRACT

As modern software systems evolve towards greater complexity, ensuring their reliable operation has become a critical challenge. Log data analysis is vital in maintaining system stability, with anomaly detection being a key aspect. However, existing log anomaly detection methods heavily rely on manual effort from experts, lacking transferability across systems. This has led to the situation where to perform anomaly detection on a new dataset, the operators must have a high level of understanding of the dataset, make multiple attempts, and spend a lot of time to deploy an algorithm that performs well successfully. This paper proposes LogCraft, an end-to-end unsupervised log anomaly detection framework based on automated machine learning (AutoML). LogCraft automates feature engineering, model selection, and anomaly detection, reducing the need for specialized knowledge and lowering the threshold for algorithm deployment. Extensive evaluations on five public datasets demonstrate LogCraft's effectiveness, achieving an average F1 score of 0.899, which outperforms the second-best average F1 score of 0.847 obtained by existing unsupervised algorithms. According to our knowledge, LogCraft is the first attempt to extract fixed-dimensional vectors as latent representations from a complete log dataset. The proposed meta-feature extractor also exhibits promising potential for measuring log dataset similarity and guiding future log analytics research.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

Log Anomaly Detection, Automated Machine Learning, Meta Learning

ACM Reference Format:
Shenglin Zhang, Yuhe Ji, Jiaqi Luan, Xiaohui Nie, Zi'ang Chen, Minghua Ma, Yongqian Sun, and Dan Pei. 2024. End-to-End AutoML for Unsupervised Log Anomaly Detection. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3691620.3695535

1 INTRODUCTION

As modern software systems continue to evolve towards greater scale and complexity, they are extensively designed for various online services and intelligent applications, requiring 24/7 uninterrupted provision of reliable services to users [24, 50]. Consequently, ensuring the smooth operation of these systems has become a critical challenge [7, 8, 14, 38]. Log data is the textual record generated during the operation of systems or applications, detailing the code execution process and the current state of the system. Analyzing log data is vital for maintaining system stability, and anomaly detection plays a key role in identifying system errors and potential risks [9, 18–20, 30, 36, 42].

Traditional log anomaly detection methods heavily rely on the involvement of operations and maintenance experts, who filter out anomalous logs by setting rules or searching for keywords. Such methods are not only costly in terms of manual labor, but also lack transfer ability, requiring customized solutions for different systems and application versions. With the advancement of machine learning technologies, numerous automated log anomaly detection schemes have been proposed [5, 12, 15, 16, 29, 33, 41, 48, 51, 53]. These schemes utilize machine learning or deep learning algorithms to identify anomalies in log data. Depending on the need for labeled data, these algorithms can be categorized into unsupervised and supervised types. Unsupervised algorithms do not require labeled anomaly data and detect anomalies by learning the patterns of normal data [12, 15, 16, 41]. In contrast, supervised algorithms need known anomaly data for model training and identify log anomalies through the classification process [29, 51, 52]. In industrial practice, system operators still tend to prefer using unsupervised approaches due to the lack of high-quality labeled data [39, 40].

*Yongqian Sun is the corresponding author.
TKL-SEHCI is short for Tianjin Key Laboratory of Software Experience and Human Computer Interaction. HL-IT stands for Haihe Laboratory of Information Technology Application Innovation. CNIC, CAS is short for Computer Network Information Center, Chinese Academy of Sciences.

ASE'24, 27 October-1 November 2024, Sacramento, California

ACM ISBN 979-8-4007-1248-7/24/10…$15.00
https://doi.org/10.1145/3691620.3695535

However, although existing unsupervised algorithms have shown good performance on certain datasets, they still face significant challenges in actual deployment. Selecting a high-performing algorithm for a specific dataset is very difficult and requires a considerable amount of time for iterative trials. Previous research findings [53] and our empirical study indicate that *there does not exist one anomaly detection algorithm that can get the best performance on different datasets*. We realize that automating the selection and deployment of algorithms to achieve stable performance is just as important as proposing new ones. Therefore, *this paper aims to develop an automated system that intelligently selects and deploys anomaly detection algorithms, and continuously enhances their performance in real-world applications, thereby reducing the time and cost associated with algorithm selection and deployment.*

In fact, selecting a well-performing algorithm for a specific dataset is very challenging. Each algorithm focuses on different data characteristics, uses varying model structures, and has distinct training objectives, leading to significant performance differences across datasets. Operators need to spend a considerable amount of time conducting multiple iterative trials. Previous research [25] shows that typical data science projects allocate 15% to 26% of their time to model selection or construction, representing a significant expenditure. Besides, some works focus on using automated machine learning (AutoML) to speed up the model selection process. Zhao and colleagues [54] have demonstrated the effectiveness of meta-learning-based model recommendation algorithms for automating outlier model selection on metric data. Therefore, we propose that leveraging AutoML methods for automating log anomaly detection could be a promising solution.

However, when automating the entire log anomaly detection process, we encountered the following challenges: **(1) Diversified datasets present challenges to feature engineering**. Currently, mainstream log anomaly detection algorithms consider template extraction as a preliminary step. The quality of template extraction and the chosen log representation methods (e.g., sequence, count) will significantly impact the final anomaly detection results. There are substantial differences between various log datasets, so obtaining good log feature representations requires substantial human involvement, setting different rules for each dataset, and conducting multiple trials. **(2) The challenge of massive hyperparameter combinations and unlabeled data for model selection and evaluation.** For each model, there are numerous variable hyperparameter values, and the combinations of these hyperparameters can reach millions or even tens of millions, making it very challenging to select an appropriate model. Additionally, model evaluation requires labeled data to compute relevant metrics, making it difficult to assess the relative performance of models when only unlabeled data is available.

In this paper, we propose LogCraft, an end-to-end unsupervised log anomaly detection framework based on AutoML, designed to accelerate model selection and deployment for unsupervised log anomaly detection. (1) To address the first challenge, we designed LogAFE, an adaptive log feature enhancement framework. LogAFE improves log template parsing quality through semantic analysis and achieves high-quality log representation without human intervention by combining multiple features and unifying feature and model optimization. (2) To address the second challenge, we first apply a particle swarm optimization algorithm to conduct hyperparameter search on the existing dataset, filtering out a selection of well-performing models as a coarse screening in the recommendation process. Then, we use a carefully designed meta-learner to identify the patterns between the dataset and model performance, enabling model recommendations on new datasets, corresponding to fine screening in the recommendation process.ll

LogCraft is systematically evaluated on five public datasets. Under highly automated conditions, LogCraft achieved an average F1 score of 0.899, surpassing the second-best performance of LogBERT, which had an average F1 score of 0.847. Our ablation experiments demonstrated the effectiveness of LogCraft's each component and explored the impact of the sole hyperparameter on LogCraft.

Our contributions are summarized as follows:

(1) To the best of our knowledge, LogCraft is the first work that focuses on automated unsupervised log anomaly detection. It is an end-to-end unsupervised log anomaly detection framework based on AutoML, enhancing the automation of anomaly detection and lowering the barrier for algorithm deployment.

(2) We designed two modules to address the challenges of automating log anomaly detection. First, we developed LogAFE to obtain high-quality log representations, thereby solving the feature engineering difficulties caused by log variations. Secondly, we adopted a meta-learning approach to tackle the model recommendation problem for unlabeled datasets. Specifically, we innovatively designed a meta-feature extractor tailored for log datasets to measure a certain similarity between them and associate it with algorithm performance.

(3) Demonstration of framework effectiveness and open source code. LogCraft has demonstrated exceptional performance, achieving an impressive average F1 score of 0.899 across five publicly available datasets. This remarkable result significantly surpasses the second-best baseline algorithm, which only achieved an F1 score of 0.847. LogCraft's ability to excel despite the challenges of smaller training sets and finer evaluation granularity underscores its robustness and effectiveness. Furthermore, in a commitment to advancing research in this field, we have open-sourced all the code of LogCraft. Our source code is available at [1].

2 BACKGROUND

2.1 Log Anomaly Detection

The purpose of log anomaly detection is to identify logs that may symbolize abnormal system activity. Specifically, the mainstream log anomaly detection process consists of four parts: log parsing, log grouping, log representation, and anomaly detection [30], as shown in Figure 1.

(1) Log parsing. Each log entry typically contains two parts: a structured part and an unstructured part. The structured part of the log depends on the log recording tool used, while the unstructured part is written by the programmer [10]. Most log anomaly detection algorithms require structured data (such as a matrix or a list of structured log counts) [18]. Therefore, converting them into structured log events through log parsing algorithms is very important. Many log parsing schemes have been proposed, such as Spell [11], Drain [17], FT-Tree [41], LogPPT [31], etc. Among them,

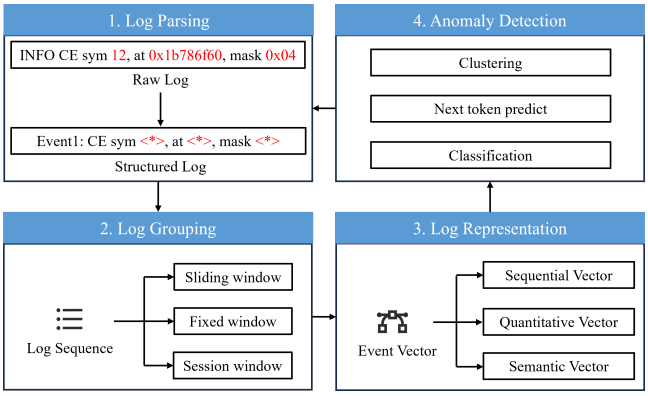

Figure 1: Pipeline of log anomaly detection approaches

Drain is used by many anomaly detection works due to its good performance on various datasets [15, 21, 30, 48, 52].

(2) Log grouping. Direct analysis of log data is not feasible because they are too large and complex. Moreover, since there is a strong correlation between the contexts of logs, the results of analyzing individual logs are often unreliable. Existing work typically uses three methods to group logs, including fixed windows, sliding windows, and session windows [18].

(3) Log representation. Machine learning models require numerical output, but logs are usually in text form. Therefore, it is necessary to extract reasonable features from logs and represent them as continuous vectors to serve as training data for models. Different anomaly detection methods will transform logs into different feature vectors. For example, DeepLog [12] uses sequences of log events as model input, LogAnomaly [41] uses both vectors of log events and event count vectors to analyze logs, and NeuralLog [29] classifies and detects logs using semantic vectors encoded by BERT.

(4) Anomaly detection. The feature data will be utilized to train a model, which may either follow a rule-based approach or be based on machine learning techniques. This model will subsequently serve in identifying anomalies within new logs, flagging any logs that appear anomalous for in-depth analysis by operators.

2.2 AutoML & Meta Learning

AutoML is an important branch of machine learning, dedicated to simplifying some of the complex and technical tasks in machine learning, allowing machine learning to achieve good results without human intervention [22]. AutoML mainly focuses on three aspects: feature engineering, model selection, and optimization algorithm selection [49]. Most AutoML algorithms require a labeled dataset to evaluate the model's performance and automatically adjust the selection of features and model structure based on the evaluation results.

Meta-learning, also known as "learning to learn", is a crucial part of automatic machine learning. Meta-learning focuses on quickly adapting to new tasks by leveraging past experiences, and it is an important subset of automated machine learning. Its principles enable the application of past knowledge to guide the completion of new tasks to a certain extent, even with unlabeled data [3]. Meta-features are defined as vectors that can characterize certain attributes of a dataset and are typically extracted from that dataset [2]. They are often employed in the automatic recommendation of machine learning algorithms and for high-performance parameter optimization. In this paper, we designed a meta-feature extractor that effectively extracts characterizations of log data. This extractor is integrated with meta-learning for recommendations in anomaly detection models.

3 EMPIRICAL STUDY

In this section, we conducted an empirical study to investigate the extent to which current log anomaly detection methods rely on human intervention. The motivation behind this study is to understand how much manual effort and domain expertise are required for effective log anomaly detection using modern techniques. Specifically, we explore the impacts of model selection and log feature engineering choices.

Model selection directly determines the performance ceiling of a log anomaly detection system [20, 53]. With the variety of neural architectures and hyperparameter configurations available, identifying the optimal model for a given scenario can be extremely challenging without extensive experimentation and domain expertise.

Effective log feature engineering is a prerequisite for accurate anomaly detection. However, this process often requires substantial human effort and domain knowledge, particularly in tasks like log parsing and feature selection, which can be dataset-specific and error-prone.

Three public datasets were used in the study, namely Blue Gene/L (BGL) [43, 55], Thunderbird [55], and the Hadoop Distributed File System (HDFS) [47, 55]. BGL and Thundbird consist of logs generated from supercomputers, while the HDFS dataset was constructed using map-reduce tasks on over 200 Amazon EC2 nodes. Due to the large size of the Thunderbird dataset, we only analyzed the first five million entries from 200 million logs.

RQ 1: How does the selection of models impact the performance of log anomaly detection? Currently, the primary neural network structures relied upon for unsupervised log anomaly detection are still RNNs [12, 41, 53] and Transformers [15]. It is worth noting that regardless of whether the neural networks are based on RNNs or Transformers, they essentially learn the features of log sequences by predicting the next token. However, different algorithms exhibit significant differences in network architecture design and hyperparameter configuration. For example, RNNs can choose different recurrent units such as LSTM or GRU, while Transformers can adjust the number of attention heads, the number of encoder/decoder layers, etc. The selection of these algorithmic parameters has a significant impact on anomaly detection performance. The performance differences brought about by different algorithmic parameters can be quite large, making it challenging for humans to search and optimize in this algorithmic space.

In this section, we define a model as a combination of algorithms and hyperparameters. This means that even if the algorithms are the same, we consider them different models if they use different hyperparameters. We conducted research using three unsupervised algorithms: DeepLog [12], LogAnomaly [41], and LogBERT [15].

Table 1: F1 scores of best and worst-performing models

Dataset	DeepLog	LogAnomaly	LogBERT
BGL	778/0.501	0.713/0.671	**0.874**/0.576
Thunderbird	0.723/0.580	0.734/0.602	**0.803**/0.672
HDFS	**0.944**/0.711	0.883/0.711	0.828/0.749

Table 2: Template counts and parsing accuracy with different settings

	Template Counts		Parsing Accuracy	
Dataset	Customized	Default	Customized	Default
BGL	**684**	1823	**0.901**	0.681
Thunderbird	**2073**	3487	**0.857**	0.590
HDFS	**35**	47	**0.990**	0.808

DeepLog employs an LSTM model to learn log sequence vectors, while LogAnomaly uses two LSTMs to learn log sequence vectors and count vectors separately. LogBERT utilizes BERT as the core model, conducting anomaly detection through masked keyword training and hypersphere volume minimization. We performed anomaly detection tasks on 54 combinations of four common hyperparameters: *window_size, stride, num_layers, hidden_size* across the three algorithms, totaling 162 models.

Observation 1: The results of log anomaly detection heavily depend on the choice of neural network architecture and the configuration of network hyperparameters. Table 1 shows the best and worst F1 scores obtained by the three algorithms on each dataset. In our experiments, we found that it is often very challenging to identify the model that performs best on different datasets. On one hand, the core structure of the algorithm and the training method can affect its anomaly detection capabilities; on the other hand, the setting of hyperparameters can also impact the model's convergence speed and learning ability.

RQ 2: How much domain expertise and manual effort is required in log feature engineering? For log segments that are out of the ordinary, it is essential to manually configure regular expressions to refine parsing accuracy [32, 56]. This task typically necessitates a comprehensive understanding of the dataset's structure and specific technical expertise. Additionally, since the log parser significantly influences the efficiency of anomaly detection methods, an increase in the number of event templates can result in reduced anomaly detection efficiency [13]. Consequently, operators should be proficient in log parsing technology and fine-tune the hyperparameters within the algorithm, thereby maintaining the template count within a reasonable limit.

Observation 2.1: Log parsing, a crucial step in log feature engineering, heavily relies on operation and maintenance experts' domain knowledge to establish appropriate parsing formats and manually configure regular expressions for anomalous log segments. Table 2 contrasts the template counts and parsing accuracy(PA) derived from both settings' parsing results. PA is the proportion of accurately parsed log messages to the total log messages. Zhu et al. [56] developed logparser, an open-source toolkit for log parsing, which encompasses various parsing algorithms and tailors regular expressions and hyperparameters to specific datasets. We utilized Drain [17] for log parsing. In the *Customized Setting* experiment, we adjusted hyperparameters and crafted regular expressions based on individual parsing outcomes to minimize errors within the algorithm. Conversely, the *Default Setting* experiment employed only Drain's standard hyperparameters and fundamental regular expressions (IPv4, IPv6, decimal, and hexadecimal numbers) for template generation. The experiment results demonstrate that failing to choose appropriate hyperparameters and set corresponding regular expressions for the template parsing algorithm can result in many logs being incorrectly parsed, significantly increasing the total number of templates obtained. Moreover, le et al. [29] discovered that current log parsing approaches may yield inaccurate parsing due to Out-Of-Vocabulary (OOV) issues and semantic misinterpretations, adversely affecting the efficacy of subsequent anomaly detection algorithms.

Observation 2.2: Apart from log parsing, feature selection also significantly affects the detection performance of the model. Landauer et al. [28] have demonstrated the necessity of selecting appropriate features for anomaly detection by applying simple rule-based algorithms to different datasets and focusing on different features. In the study, using event sequences as the target feature for anomaly detection yielded an F1 score of 53.9 on HDFS while using count vectors as the feature yielded an F1 score of 56.0. When both features were combined for detection, the F1 score increased to 72.0. However, combining these two features on Thunderbird and BGL resulted in a performance decline.

4 APPROACH

To minimize the need for manual intervention in log anomaly detection and enhance algorithm deployment, we introduce LogCraft. This innovative, end-to-end, unsupervised log anomaly detection framework aims to automate the process significantly. LogCraft excels by incorporating feature engineering and model selection within its architecture, thereby facilitating superior anomaly detection in log datasets with minimal human oversight. In this study, the dataset used to construct the meta-learner is termed the *support set*, while the dataset used for actual anomaly detection is the *target set*. LogCraft comprises three main components: LogAFE for feature enhancement, Meta-Learner Construction for dynamic learning, and a Model Preparation segment. Figure 2 visually outlines the systematic workflow of LogCraft, highlighting the integration and interaction of these components.

4.1 LogAFE: Adaptive Log Feature Enhancement

The robustness of data cleansing and preparation plays a pivotal role in fortifying the learning proficiency of automated models [22]. Positioned at the forefront of AutoML challenges, LogAFE is designed to tackle the complexities of log feature engineering observed in empirical research, focusing on the adaptive extraction of salient features from textual logs.

Laying the groundwork, we commence with employing the Drain algorithm [17] for the initial log parsing phase, from which we obtain a rudimentary set of parsing results. Since LogCraft is designed

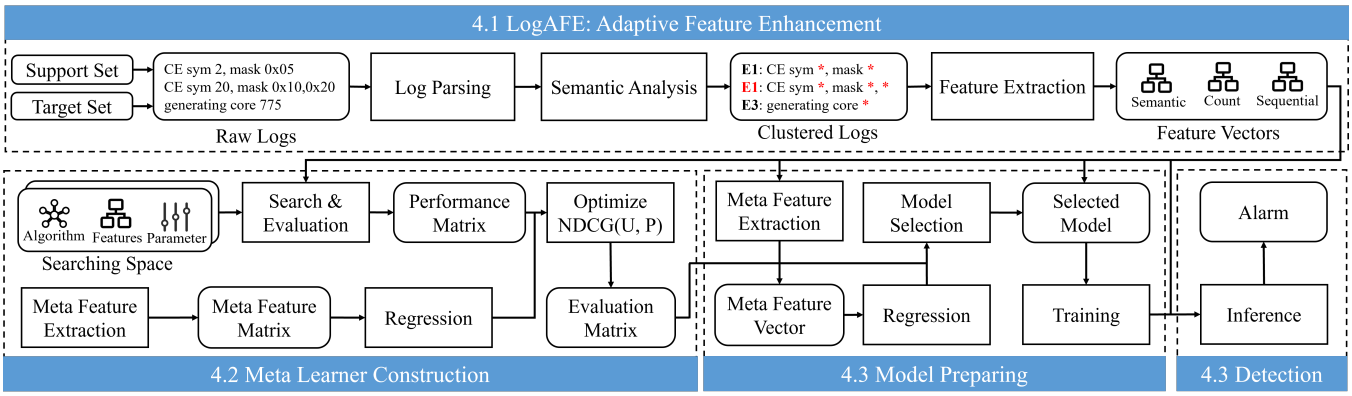

Figure 2: Anomaly detection pipeline of LogCraft

for autonomous operation, Drain is used with its default hyperparameters and predefined regular expressions. In datasets with diverse log types, this default setup may generate many redundant templates due to incorrect classification of log entries. This issue mainly arises from three types of parsing errors: logs with variable-length variables being split into multiple templates, the lack of specific regular expressions to filter out unique system strings, and traditional log parsing algorithms ignoring semantic information, leading to different templates for logs with identical meanings but slightly different wording.

To address these challenges, we proceed with semantic rectification of the initial templates. Initially, we will remove any placeholders and special symbols that represent variables and regular expressions from each parsed template. Then we encode each log template using the pre-trained model all-MiniLM-L6-v2 from the sentence-transformers repository [44, 45]. This model, fine-tuned on a billion sentences, generates 384-dimensional semantic vectors for each input statement, making it suitable for semantic similarity analysis.In line with canonical semantic analysis practices, we utilize cosine similarity amongst the vectors to delineate template akinness. The encoded log templates undergo a clustering algorithm, leveraging the union-find strategy to amalgamate analogous templates. Herein, the most prevalent template within each collective emerges as the archetype. This stratagem can effectively mitigate the negative impact of the three aforementioned errors on template parsing, ensuring that the feature engineering module achieves excellent performance even in a fully automated setting. Moreover, reducing the number of templates will significantly enhance the efficiency of subsequent model training and detection processes.

Then, we map the merged templates to unique template IDs, transforming the log template sequence into a log ID sequence, serving as the input log event vectors for subsequent modeling. Furthermore, we generate count vectors based on the frequency of each template within a specified window, providing an additional feature for anomaly detection. These vectors, in conjunction with the magistral semantic representations from Sentence-BERT, afford LogAFE with a multi-faceted feature amalgam, comprising event, count, and semantic vectors, thereby assembling a formidable feature combination search space. Empirical study on different datasets indicate that the reasonable combination of various types of features can significantly enhance anomaly detection performance. By enveloping feature assemblies, algorithmic selection, and hyperparameter tuning within a meta-learning paradigm, LogAFE autonomously steers towards the most optimal model configurations for unfamiliar datasets. Culminating in a unified optimization conceptualization, this approach amplifies LogCraft's generalization aptitude. Our forthcoming sections will delve into meta-feature learning and algorithm selection, canvassing through the assembled search space, to procure configurations that assure illustrious performances across different datasets.

4.2 Meta Learner Construction

The Meta Learner Construction phase in LogCraft is crucial as it forms the foundation for recommending optimal models for anomaly detection. This phase involves a meticulous offline training process designed to understand and leverage the relationship between different models and their effectiveness on various datasets. The final meta-learner consists of two parts: a regressor and a trained matrix

Table 3: Selected statistical features and their meanings

Statistical Feature	Feature Meaning
Maximum, Minimum	Frequent occurrence of templates
Array Length	Number of templates
Variance	Dispersion of templates
Skewness	Asymmetry of template distribution
Kurtosis	Sharpness of template distribution

Meta feature extraction. The core challenge in recommending models for unlabeled log datasets lies in uncovering the relationship between model performance and data characteristics. Effective feature extraction must not only represent the vast dataset but also ensure these features possess good transferability. LogCraft addresses this challenge by innovatively designing a log meta-feature extractor, which extracts meta-feature vectors that encapsulate the

patterns or characteristics of an entire log dataset. To our knowledge, LogCraft is pioneering in characterizing entire log datasets with finite-dimensional vectors for data correlation analysis. Specifically, the log meta-feature vector extracted by LogCraft includes two parts: statistical meta-features and model-based meta-features.

Statistical meta-features capture the distribution information of the log data. We focus on template counts and template sequences, which are highly robust and universal features. The selected statistical features for this vector and their corresponding meanings are detailed in Table 3. We employ a two-level statistical feature calculation method. Initially, we compute statistical features for each log template to capture its distribution characteristics. Subsequently, we calculate global statistical features for the entire matrix to extract characteristics that represent the overall log flow. Furthermore, we introduce the template distribution proportion feature, considering the importance of the distribution and frequency of log templates in distinguishing between normal and anomalous log sequences. Lastly, during the meta-feature extraction process, we conduct the Kwiatkowski–Phillips–Schmidt–Shin(KPSS) test [27] on each sequence, incorporating the resulting data as part of the statistical meta-features representing sequence stability. Specifically, in the process of applying the KPSS test for meta-feature extraction, we compute the KPSS statistic, p-value, and lag order for each sequence. These metrics collectively provide a comprehensive description of the statistical features of the sequences.

In the field of meta-learning, selecting appropriate model-based meta-features is considered a key step in building efficient meta-learning models [54]. These features focus on extracting key information from the dataset through simple and fast algorithms. In this study, we selected structural information generated by two unsupervised algorithms—Isolation Forest(IForest) [35] and LogCluster [34] as meta-features. IForest is a tree-based anomaly detection algorithm that efficiently identifies outliers in data through an isolation mechanism. We use parameters such as the number of decision trees and splitting thresholds as meta-features. LogCluster is a clustering-based log anomaly detection algorithm that identifies problems and abnormal patterns in log data through automatic clustering. We select features related to the number of clusters, the range of cluster sizes, and inter-cluster distances. This step yields an 8-dimensional vector of model-based meta-features.

Ultimately, our designed meta-feature extractor can extract a 59-dimensional meta-feature vector for each log dataset, which is used for subsequent meta-feature learning algorithms. Moreover, this extractor can also guide the assessment of log dataset similarity and the analysis of relationships between datasets. Concatenate the meta-feature vectors generated from all logs in the support set to form a matrix M of size $n*d$, where n is the number of log sets in the support set, and d is the dimension of the extracted meta-features.

The overarching goal of the meta-learner construction phase is to discover a high-performing algorithm that can select the optimal model from the search space based on the meta-feature vector extracted from new datasets. This phase is designed to bridge the gap between the characteristics of the data and the performance of various models, thus ensuring effective anomaly detection.

Inspired by the work of Zhao et al. [54], our approach employs a model-based collaborative filtering algorithm to learn the mapping relationship between meta-features and models. The process is meticulously organized as follows:

Searching space initialization and model performance evaluation. During this phase, we define the search space consisting of various models. Each model is a combination of features, algorithms, and hyperparameters. Specifically, the features include event vectors, count vectors, and semantic vectors, as well as their combinations. The algorithms encompass four unsupervised log anomaly detection algorithms: DeepLog [12], LogAnomaly [41], CNN [6],and LogBERT [15].

To ensure that our framework performs well on different datasets, we initialized the hyperparameter search space across a wide range. The range of each hyperparameter and the corresponding models are shown in Table 4, resulting in a total of 11,136,000 algorithm and hyperparameter combinations.

To select models that are likely to perform well on the target dataset under the huge searching space and reduce the cost of training the meta-learner, we first run the Particle Swarm Optimization algorithm (PSO) [26] on the labeled support set. PSO simulates the birds in a flock by designing massless particles. Each particle has two properties: velocity and position. Velocity represents the speed of movement, while position indicates the direction of movement. Each particle individually searches for the optimal solution in the search space and records it as the current personal extremum. This personal extremum is then shared with other particles in the entire swarm to find the optimal personal extremum, which becomes the current global optimal solution for the entire swarm. All particles in the swarm adjust their velocity and position based on their own found current personal extremum and the current global optimal solution shared by the entire swarm. For each algorithm, we select the top 1000 hyperparameter combinations based on their performance on each support set and use these models as the candidate set for the meta-learner. After removing those identical models, we obtained a candidate set of models with a size of 7840. The subsequent task is to recommend the most suitable model on a new unlabeled dataset.

Next, in order to provide prior knowledge for subsequent recommendation algorithms, we will train and test the model selected by the PSO algorithm using support set to evaluate the model's performance. We use F1 score as the evaluation metric. These F1 scores form a matrix P of size $n \times m$, where n represents the number of existing datasets, and m is the number of models. This setup allows us to capture the relationship between different models and their performance across various datasets.

Collaborative filtering for model recommendation. The goal of the collaborative filtering algorithm is to recommend suitable models by analyzing patterns in the matrix P. We initialize a matrix V of size $m * d$ using a normal distribution and use matrix M to initialize another matrix U.

Our objective is to minimize the difference between the model evaluation matrix P and the reconstructed matrix UV^T. We use Normalized Discounted Cumulative Gain (NDCG) as the loss function to assess the ranking differences between the two matrices:

$$Loss = 1 - NDCG(P, UV^T) \tag{1}$$

Table 4: Parameter Search Spaces and Their Corresponding Models

Parameter	Search Space	Corresponding Models	Description
epochs	2, 4, 6, 8, 10	ALL	Number of passes through the training data.
hidden_size	16, 32, 54, 128, 256, 512, 768, 1024, 2048, 4096	ALL	Units in each hidden layer.
num_layers	2, 4, 6, 8, 10, 12, 14, 16, 18, 20	ALL	Total number of layers in the model.
window_size	5, 10, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200	ALL	Size of the input window.
use_quantitive	0, 1	ALL	Use quantitative features (0 = No, 1 = Yes).
use_semantic	0, 1	ALL	Use semantic features (0 = No, 1 = Yes).
use_attention	0, 1	DeepLog, LogAnomaly	Apply attention mechanisms (0 = No, 1 = Yes).
num_directions	1, 2	DeepLog, LogAnomaly	Directions for RNNs (1 = uni, 2 = bi).
embedding_dim	16, 32, 64, 128, 256, 512, 1024, 2048	DeepLog, LogAnomaly, LogBERT	Dimensionality of embeddings.
n_head	2, 4, 6, 8, 10, 12, 14, 16, 18, 20	LogBERT	Number of attention heads.
latent_size	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1	CNN	Size of latent representation.
kernel_size	3×3, 5×5, 7×7, 9×9	CNN	Size of the convolutional kernel.
pooling_size	2×2, 3×3, 4×4, 5×5, 6×6, 7×7, 8×8, 9×9	CNN	Size of pooling layer.

NDCG is a commonly used ranking evaluation metric that measures the consistency between predicted and actual rankings. We use NDCG@3 in our framework, which reflects the degree of consistency between the actual ranking and the ideal ranking when only the top 3 entries of the sorting results are considered. The definition of NDCG@k is as follows:

$$NDCG@k = \frac{DCG@k}{IDCG@k} \tag{2}$$

DCG@k measures the overall relevance of items in the list, taking into account their positions. As the position decreases, the value decreases, which is represented by a discount factor. DCG@k is calculated as the sum of the multiplication of the relevance score of each item and the reciprocal of the logarithm of its position.

$$DCG@k = rel_1 + \sum_{i=2}^{k} \frac{rel_i}{\log_2(i+1)} \tag{3}$$

IDCG@k is the max DCG value that can be obtained when all items are arranged in order of relevance, preset on the same relevance score. When calculating IDCG@k, the most relevant item is assumed to be at the top of the list. Here, $|REL_k|$ is the set consisting of the top-k results after sorting rel_i from highest to lowest.

$$IDCG@k = \sum_{i=1}^{|REL_k|} \frac{2^{rel_i}-1}{\log_2(i+1)} \tag{4}$$

In our scenario, each row of P corresponds to a dataset, representing the actual ranking of different models on that dataset (based on F1 scores); UV^T represents the predicted ranking of models. By minimizing this loss function, we can learn the optimal V, making UV^T as close as possible to the actual model evaluation matrix P.

After the aforementioned training process, we obtain well-trained matrices U and V. Next, we will use a random forest regressor to learn the mapping from matrix M to matrix U, which will later be applied to transform meta-feature vectors in the target set similarly.

4.3 Model Preparing and Detection

The objective of this step is to utilize the meta-learner to recommend a well-performing model for the unlabeled target set. The process is organized as follows: First, we perform LogAFE on the target set to obtain structured files and their event vectors, count vectors, and semantic vectors. Next, we input the structured files into the meta-feature extractor to get a meta-feature vector m representing the data set. We use the random forest regressor trained in the meta-learner to transform the vector m, resulting in a $1\times d$ vector u. We then take the dot product of vector u and matrix V to get a $1\times m$ vector q.

$$q = f(m)\cdot V^T \tag{5}$$

Vector q is the predicted score for each model in the target set, where each column value represents the potential evaluation score that the corresponding model in the search space may achieve on the data set, i.e., the F1 score. Lastly, we select the model with the highest score to train and test on the target set.

5 EVALUATION

In this section, we evaluate LogCraft with several experiments to answer the following research questions(RQs).

RQ3: How effective is LogCraft in unsupervised log anomaly detection?

RQ4: How effective are the main components of LogCraft?

RQ5: How do hyperparameter settings affect the performance of LogCraft?

5.1 Experimental Design

5.1.1 Datasets. we evaluated the effectiveness of LogCraft on five publicly available datasets: HDFS, Blue Gene/L (BGL), ThunderBird, Spirit, and Liberty [43, 47, 55]. The HDFS dataset was collected from a hadoop distributed file system running on the Amazon EC2 platform. The BGL dataset was collected from the Blue Gene/L supercomputer at Lawrence Livermore National Labs. ThunderBird, Spirit, and Liberty were collected from two real-world supercomputers at Sandia National Labs. Due to the large size of these three datasets, we randomly sampled 5,000,000 log entries from each dataset for evaluation.

In the HDFS dataset, each log entry contains a *block_id* that symbolizes the session to which the log belongs. We use it as a unique identifier to group logs. For the remaining log data, we group them based on time intervals. We use a sliding window with a window length of 60 seconds and a step size of 30 seconds to segment log

Table 5: Detailed information of the datasets

Dataset	Category	Messages	Anomalies
HDFS	Distributed system	11,175,629	16,838
BGL	Supercomputer	4,747,963	348,460
ThunderBird	Supercomputer	5,000,000	76,130
Spirit	Supercomputer	5,000,000	764,890
Liberty	Supercomputer	5,000,000	1,814,386

sessions. Compared to studies that set the session length to several minutes or half an hour [10, 15], this approach allows for a more rigorous evaluation of the model's anomaly detection performance at a finer granularity. Consequently, both the F1 scores of this method and the baseline experimental model will experience some decline. The sources, total log entries, and the number of anomalous log entries for each dataset are summarized in Table 5.

5.1.2 Evaluation Metrics. We adopted the commonly used Precision, Recall, and F1 Score to evaluate the log anomaly detection effect of LogCraft. These three metrics are widely used in log anomaly detection tasks to comprehensively evaluate the model's ability to distinguish between normal and abnormal data [6, 10, 12, 34, 41, 51, 53, 55], and the definitions of these three metrics are as follows:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

5.2 How Effective Is LogCraft?

This research RQ assesses the performance of LogCraft in log anomaly detection. In evaluating LogCraft's performance on a specific dataset, it was designated as the target set, with the other four datasets serving as the support set for constructing the knowledge base. In fact, only one construction of the meta-learner is required, followed by masking the corresponding rows of the target set in the matrix. For the target set, we randomly selected 10% of the normal logs for model training, with the remainder normal logs and all abnormal logs used for anomaly detection. The only hyperparameter that needs to be set in this framework is the similarity threshold for merging templates during template semantic analysis(s_th) in the feature engineering module, which was set at 0.9. We conducted three experiments starting from the construction of the meta-learner and took the average as the final result.

We compared the performance of LogCraft with six unsupervised log anomaly detection baselines: PCA [46], IM [37], DeepLog [12], LogAnomaly [41], CNN [6], LogBERT [15], and LogTAD [16]. PCA and IM are based on traditional machine learning algorithms; DeepLog and LogAnomaly are based on LSTM deep learning algorithms; CNN is a type of deep learning architecture specifically designed for processing structured grid data; LogBERT is a BERT-based anomaly detection method; LogTAD is an unsupervised log transfer learning algorithm. For the first six algorithms, we only trained using the target set. For LogTAD, we adopted the bilateral generalization experiment setup from its paper with BGL and ThunderBird. For other target sets, we used a meta-feature extractor to find the dataset with the highest similarity to its meta-feature vector from the support set for transfer experiments. Specifically, we selected BGL as the source domain for Liberty and HDFS, and Liberty as the source domain for Spirit. In the source domain, we randomly sampled 10% of the normal data for LogTAD training, with training and detection in the target domain consistent with other experiments. In order to reduce the impact of random factors, we conducted three experiments and took the average of the final results.

The experimental results, as shown in Table 6, demonstrate the effectiveness of LogCraft in achieving the highest F1 scores on the BGL, Spirit, HDFS, and ThunderBird datasets. Specifically, for the detection tasks of BGL and Liberty, LogCraft recommended a model based on LogBERT; for the detection tasks of HDFS and Spirit, LogCraft selected a model based on DeepLog; and for the ThunderBird dataset, LogBERT opted for a CNN model with adjusted parameters for detection. From the results, it is evident that no single anomaly detection method performs well across all datasets. PCA, as a linear dimensionality reduction technique, is adept at capturing linear relationships within the dataset. On the other hand, IM's detection performance relies on prior feature mining, and these two algorithms may exhibit good performance on certain datasets but exhibit significant fluctuations when faced with diverse and complex data.

DeepLog, LogAnomaly, LogBERT, CNN, and LogTAD are deep learning-based algorithms with more complex architectures and higher parameter capacities. They can extract higher-order or nonlinear features directly from the raw dataset, which contributes to their improved robustness and overall performance. However, the most suitable features for detection and the optimal hyperparameters for learning vary across different datasets. Consequently, LogCraft, which incorporates high-quality feature engineering and model recommendations, ultimately demonstrates superior robustness and achieves the highest average F1 score.

LogTAD performs best on the Liberty dataset and achieves good results on the Spirit dataset, but its performance is poorer on other datasets. We believe that the effectiveness of such transfer learning algorithms is influenced by latent relationships between different datasets, and currently, there is no effective criterion for selecting the most suitable dataset. This finding underscores the challenges faced in current cross-domain work. Overall, LogCraft, which recommends models based on past experience rather than transferring from a single dataset, achieves the best average performance, demonstrating the algorithm's robustness.

Notably, LogCraft exhibits significant performance improvements on the BGL and ThunderBird datasets, which are characterized by complex log types and lower template parsing accuracy. In summary, LogCraft automates log feature engineering, model recommendation, and anomaly detection without manual intervention, achieving a high level of automated log anomaly detection. Across the six datasets, it achieves an average F1 score of 0.899 without human intervention.

Table 6: Precisions, Recalls, and F1 Scores of different methods on different datasets

Dataset		PCA	IM	DeepLog	LogAnomaly	CNN	LogBERT	LogTAD	LogCraft
BGL	Precision	0.445	0.822	0.690	0.639	0.711	0.790	0.723	**0.854**
	Recall	0.895	0.702	0.890	0.806	0.650	**0.978**	0.581	0.935
	F1 Score	0.594	0.757	0.778	0.713	0.679	0.874	0.644	**0.893**
HDFS	Precision	0.481	0.502	0.924	0.834	0.706	0.969	0.788	**0.983**
	Recall	0.881	**1**	0.965	0.934	**1**	**1**	0.932	**1**
	F1 Score	0.622	0.668	0.944	0.883	0.828	0.953	0.854	**0.992**
ThunderBird	Precision	0.547	0.468	0.633	0.626	**0.792**	0.715	0.021	**0.756**
	Recall	0.462	0.619	0.843	0.886	0.694	0.915	0.234	**0.995**
	F1 Score	0.501	0.533	0.723	0.734	0.740	0.803	0.038	**0.859**
Spirit	Precision	**0.900**	0.564	0.652	0.591	0.822	0.724	0.702	0.890
	Recall	0.681	0.901	0.798	0.848	0.647	0.934	0.843	**0.944**
	F1 Score	0.740	0.694	0.718	0.696	0.724	0.816	0.766	**0.916**
Liberty	Precision	0.528	0.742	0.795	0.648	0.543	0.680	**0.904**	0.741
	Recall	0.728	0.646	0.833	0.894	0.925	0.941	**0.991**	0.986
	F1 Score	0.612	0.690	0.814	0.752	0.684	0.790	**0.943**	0.846

5.3 How Effective are the Main Components in LogCraft?

We explore the effects of key components in LogCraft. Specifically, through ablation experiments on LogAFE, hyperparameter search algorithm and the database construction module, we compare the performance changes of the model in anomaly detection tasks.

For LogAFE, we conducted the following ablation experiments. First, we eliminated the task of merging template semantics and directly extracted feature vectors for model training. The experimental results are shown in Figure 3. The effectiveness of this approach was validated in terms of performance and efficiency. Upon removing the semantic analysis of the template, the accuracy of template parsing declined, causing some logs in the test set to be misclassified. As a result, these logs appeared as out-of-vocabulary templates during detection, causing false alarms by the model. Concurrently, due to the increase in the number of templates, the model's execution time for anomaly detection tasks also increased. The experimental results showed that in most datasets, performing template merging via semantic analysis yielded better detection results and reduced the required time for algorithm execution. The final score and running time did not improve, because only 44 templates existed in the initial parsing of the HDFS dataset and no similar templates were merged during the merging process.

To study the impact of different hyperparameter algorithms on the quality of the candidate set for meta-learning models, we compared three hyperparameter selection algorithms: Random Search (RS) [4], Genetic Algorithm (GA) [23], and Particle Swarm Optimization (PSO) [26]. Our experiment utilized a diverse collection of 11,136,000 pre-trained models. The dataset was split into 10% training and 90% test sets. We employed F1-score as the performance metric. To manage computational constraints while maintaining result reliability, each algorithm was executed 3 times. RS was configured to sample 2,000 models. GA utilized a population size of 2,000 over 100 generations, with crossover and mutation rates set at 0.8 and 0.1, respectively. Similarly, PSO was implemented with 2,000 particles evolving over 100 iterations, using an inertia weight of 0.7 and both cognitive and social learning factors of 1.5. The final experimental results are shown in the Table 8. PSO, while having a significantly shorter total runtime compared to GA, exhibits performance similar to that of GA.

Table 7: Comparison of the overhead and performance of Random Search (RS), Genetic Algorithm (GA), and Particle Swarm Optimization (PSO).

Algorithm	Time Cost(s)	Avg F1Score
RS	22,423	0.781
GA	52,169	0.904
PSO	30,082	0.899

The experimental results indicate that although the Random Search Algorithm has the lowest overhead, its average performance in generating candidate sets is relatively poor, which limits the performance ceiling of subsequent model recommendations. Both the Genetic Algorithm and Particle Swarm Optimization yield similar performance in their candidate sets; however, the time overhead of the Genetic Algorithm is approximately 70% higher than that of the latter. Therefore, considering the balance between performance and time overhead, we chose Particle Swarm Optimization as the hyperparameter optimization algorithm to be used.

To study the impact of focused features on the model's anomaly detection performance, we tried to use only one type of feature vector for anomaly detection of the log dataset, rather than adding their combination to the search space. We compared changes in the model's anomaly detection F1 score as shown in Figure 4. Among these, *Evt Only* indicates that only event vectors are used, *Cnt Only* signifies the use of quantity vectors alone, and *Sem Only* implies the exclusive use of semantic vectors.

To validate the impact of knowledge base construction on the effectiveness of model recommendations, we compared the existing collaborative filtering-based recommendation algorithm with several alternative recommendation methods in terms of their performance in cold start scenarios. The experimental results are shown

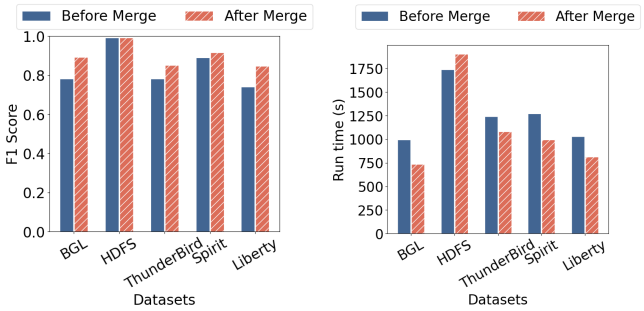

Figure 3: Comparison of F1 score and runtime before and after merging templates

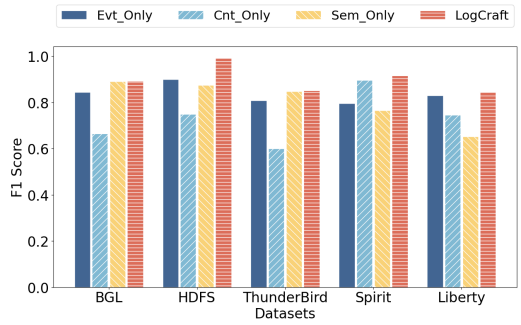

Figure 4: Comparison of F1 score for anomaly detection using only a single feature

in Table 8. The alternative recommendation algorithm schemes include the following.

Global Best. This method does not rely on meta-features. Instead, it simply selects the model with the best average performance across all training datasets for recommendation. The strategy is based on the assumption that algorithms performing well on existing datasets will also exhibit excellent performance on new datasets. However, it overlooks the potential differences between datasets. The globally optimal model may not always be the best choice.

ArgoSmArT. ArgoSmArT utilizes meta-features to characterize datasets. It selects the dataset from the training set that is closest to the target dataset in terms of meta-features for model recommendation. The method assumes a linear relationship between dataset similarity and model performance, implying that datasets with similar meta-features will exhibit similar model performance. However, the relationship between dataset similarity and model performance may not strictly follow a linear pattern.

LinearRegression. Linear regression is a predictive analysis technique that seeks to find a linear relationship between variables by minimizing the differences between predicted and actual values. It is suitable for scenarios where the relationship between variables is close to linear, but may not perform well if the data has nonlinear characteristics or complex multivariate interactions.

RandomForest. Random forest regression is an ensemble learning method based on decision trees, which improves prediction accuracy by building multiple decision trees and outputting their average prediction. It is effective for handling high-dimensional data and nonlinear relationships but can become computationally intensive when dealing with very large datasets and is not as intuitive as linear regression when it comes to model interpretation.

Theoretical optimum. We manually reviewed the evaluation metrics of the models on this dataset and identified the model that demonstrated the best actual performance. This score represents the performance ceiling of all models.

In our experiments, the collaborative filtering algorithm demonstrated the best recommendation performance. We observed that although linear regression and random forest regression show strong learning capabilities, they are prone to overfitting issues in meta-feature training or being influenced by extreme values. Moreover, when facing the cold start problem, which involves recommending models for unevaluated log data, collaborative filtering tends to perform better by analyzing the interaction patterns between users and items. Furthermore, LogCraft recommended the best-performing models within the search space for two datasets, and the models it recommended for the other three datasets also approached the theoretical optimum. The meta-learner trained using collaborative filtering achieved an average NDCG@3 score of over 0.9.

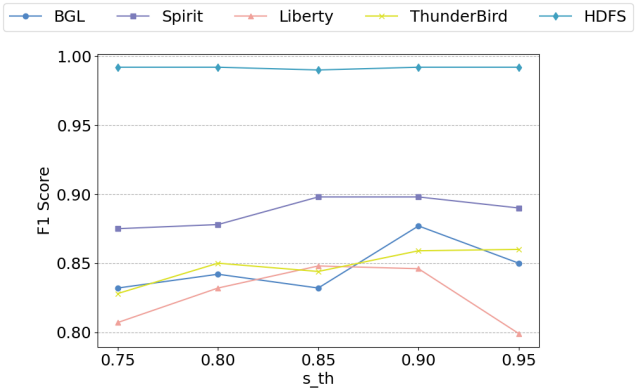

Figure 5: Impact of *s_th* on the performance of LogCraft

5.4 How do hyperparameter settings affect the performance of LogCraft?

This research question explores the impact of hyperparameters on the performance of LogCraft. LogCraft is designed as a log anomaly detection framework that minimizes manual intervention, including the reliance on manually tuning hyperparameters. Therefore, LogCraft has only one hyperparameter that needs adjustment: the *similarity threshold*(*s_th*) used for template semantic analysis. This hyperparameter directly affects the merging strength of templates, thereby influencing the quantity and quality of the final templates obtained. To study its effect on the framework's performance, we varied its value from *0.75* to *0.95* and recorded the changes in the F1 scores for anomaly detection.

The experimental results are shown in Figure 5. Overall, the performance of LogCraft remains stable, although the impact of changing *s_th* varies across different datasets. For instance, the

Table 8: F1 scores of the models with different recommendation algorithms

Method	Global Best	ARGOSMART	RandomForest	LinearRegression	LogCraft	Theoretical optimum
BGL	0.603	0.580	**0.891**	0.846	0.877	**0.891**
HDFS	0.793	0.930	0.890	**0.992**	**0.992**	**0.992**
Liberty	0.822	0.803	0.820	0.803	**0.846**	**0.846**
Spirit	0.662	0.575	0.915	0.785	0.898	**0.916**
Thunderbird	0.459	0.854	0.543	0.720	**0.859**	**0.859**
Average	0.668	0.748	0.811	0.829	**0.894**	-

Thunderbird dataset shows significant variations in F1 scores (highest 0.853, lowest 0.799), likely because Thunderbird logs are generally more complex with a higher number of templates. In contrast, for the HDFS dataset, which has the fewest templates, changes in the hyperparameter have a minimal effect. When *s_th* is set to 0.9, LogCraft achieves the optimal F1 scores across multiple datasets.

6 DISCUSSION

6.1 Lessons Learned

Through the development and evaluation of LogCraft, several valuable lessons were learned that highlight the complexities and opportunities in the field of unsupervised log anomaly detection using automated machine learning (AutoML) techniques.

Log Data Representation Matters. LogCraft implements LogAFE to enhance the feature representation of each dataset and find suitable combinations. The evaluation results demonstrate the effectiveness of this approach. Since anomaly characteristics exhibited by datasets from different sources can vary greatly, anomaly detection algorithms that focus on only one or a few features may struggle to achieve consistent performance across all datasets. When the capabilities of deep learning models reach saturation, data representation can easily become a limiting performance factor.

Meta Feature Potentials Log Analytics. The meta-feature extractor proposed in LogCraft is an innovative point, as it can generate fixed-dimensional vector representations for entire log datasets. Our experiments show that this representation can guide model recommendations and measure the similarity between log datasets. This opens up a new research direction in log analysis, namely, how to conduct data association analysis and knowledge transfer based on the "features" of entire datasets. Therefore, LogCraft not only addresses practical issues in log anomaly detection but also provides valuable exploration for meta-learning in this field.

7 RELATED WORK

This section introduces existing unsupervised log anomaly detection algorithms and MetaOD. In this paper, DeepLog, LogAnomaly, and LogBERT are used as the base algorithms for the search space.

DeepLog. Du et al. [12] utilize the LSTM model to model the sequence of log events, determining the presence of anomalies in the log sequence by predicting the template of the next log entry.

LogAnomaly. Meng et al. [41] proposed LogAnomaly, which utilizes log vectors and count vectors as inputs to train two separate LSTM models. Additionally, they introduced template2vec, a novel and effective method for extracting semantic information from log templates by considering synonyms and antonyms.

CNN. Chen et al. [6] proposed a novel semantic embedding representation method that allows for learning information from normal log sequences using a CNN network. They demonstrated that the model designed in this way can achieve excellent anomaly detection results even when trained with a small batch of normal data.

LogBERT. Guo et al. [15] utilize the BERT model to learn patterns in normal log sequences through two unsupervised learning tasks, which allow LogBERT to capture the underlying patterns in normal log sequences and detect deviations from those patterns.

LogTAD. Han et al. [16] proposed a transfer learning model LogTAD, which utilizes an LSTM network to model log sequences from two different systems. By employing domain adversarial techniques, the model maps the log sequences from the source and target domains onto the same hypersphere with a similar distribution.

MetaOD. Zhao et al. [54] innovatively established a connection between the model selection problem in outlier detection and the cold start problem in collaborative filtering, and for the first time, proposed an algorithm for automatic outlier detection model selection based on meta-learning—MetaOD.

8 CONCLUSION

Log anomaly detection is a key technology for maintaining the high availability of system services. In our empirical research, we have demonstrated the high dependency of existing detection methods on manual expertise. This paper introduces LogCraft: an end-to-end unsupervised log anomaly detection framework based on AutoML, designed to lower the barriers and domain-specific knowledge requirements for deploying algorithms in practical applications, and it has shown good performance on five public datasets with an average F1 score of 0.899, surpassing existing unsupervised detection algorithms. To our knowledge, LogCraft represents the initial effort to derive fixed-dimensional vectors as latent feature representations from an entire log dataset. Moreover, the meta-feature extractor we propose also demonstrates significant promise for assessing dataset similarity and advancing research in the field of log analytics.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (62272249, 62302244, 62072264).

REFERENCES

[1] [n. d.]. LogCraft. https://anonymous.4open.science/r/LogCraft-54D8/. Accessed: 2024-06-08.

[2] Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís PF Garcia, Jefferson T Oliva, and André CPLF De Carvalho. 2020. MFE: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research* 21, 111 (2020), 1–5.

[3] Maroua Bahri, Flavia Salutari, Andrian Putina, and Mauro Sozio. 2022. AutoML: state of the art with a focus on anomaly detection, challenges, and research directions. *International Journal of Data Science and Analytics* 14, 2 (2022), 113–126.

[4] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of machine learning research* 13, 2 (2012).

[5] Christophe Bertero, Matthieu Roy, Carla Sauvanaud, and Gilles Trédan. 2017. Experience report: Log mining using natural language processing and application to anomaly detection. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 351–360.

[6] Hao Chen, Ruizhi Xiao, and Shuyuan Jin. 2021. Unsupervised Anomaly Detection Based on System Logs.. In *SEKE*. 92–97.

[7] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2024. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024*. ACM, 674–688. https://doi.org/10.1145/3627703.3629553

[8] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. 2020. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1487–1497.

[9] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R Lyu. 2021. Experience report: Deep learning-based system log analysis for anomaly detection. *arXiv preprint arXiv:2107.05908* (2021).

[10] Zhuangbin Chen, Jinyang Liu, Wenwei Gu, Yuxin Su, and Michael R Lyu. 2021. Experience report: Deep learning-based system log analysis for anomaly detection. *arXiv preprint arXiv:2107.05908* (2021).

[11] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.

[12] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[13] Ying Fu, Meng Yan, Zhou Xu, Xin Xia, Xiaohong Zhang, and Dan Yang. 2023. An empirical study of the impact of log parsers on the performance of log-based anomaly detection. *Empirical Software Engineering* 28, 1 (2023), 6.

[14] Bernd Grobauer and Thomas Schreck. 2010. Towards incident handling in the cloud: challenges and approaches. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*. 77–86.

[15] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks (IJCNN)*. IEEE, 1–8.

[16] Xiao Han and Shuhan Yuan. 2021. Unsupervised cross-system log anomaly detection via domain adaptation. In *Proceedings of the 30th ACM international conference on information & knowledge management*. 3068–3072.

[17] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*. IEEE, 33–40.

[18] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)* 54, 6 (2021), 1–37.

[19] Shilin He, Xu Zhang, Pinjia He, Yong Xu, Liqun Li, Yu Kang, Minghua Ma, Yining Wei, Yingnong Dang, Saravanakumar Rajmohan, and Qingwei Lin. 2022. An empirical study of log analysis at Microsoft. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*. ACM, 1465–1476. https://doi.org/10.1145/3540250.3558963

[20] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 207–218.

[21] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2020. Loghub: A large collection of system log datasets towards automated log analytics. *arXiv preprint arXiv:2008.06448* (2020).

[22] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-based systems* 212 (2021), 106622.

[23] John H Holland. 1992. Genetic algorithms. *Scientific american* 267, 1 (1992), 66–73.

[24] Pengxiang Jin, Shenglin Zhang, Minghua Ma, Haozhe Li, Yu Kang, Liqun Li, Yudong Liu, Bo Qiao, Chaoyun Zhang, Pu Zhao, Shilin He, Federica Sarro, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. Assess and Summarize: Improve Outage Understanding with Large Language Models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*. ACM, 1657–1668. https://doi.org/10.1145/3611643.3613891

[25] Kaggle. 2018. 2018 Kaggle Machine Learning and Data Science Survey. https://www.kaggle.com/kaggle/kaggle-survey-2018. Accessed: 2024-05-13.

[26] James Kennedy and Russell Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, Vol. 4. ieee, 1942–1948.

[27] Denis Kwiatkowski, Peter CB Phillips, Peter Schmidt, and Yongcheol Shin. 1992. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of econometrics* 54, 1-3 (1992), 159–178.

[28] Max Landauer, Florian Skopik, and Markus Wurzenberger. 2023. A Critical Review of Common Log Data Sets Used for Evaluation of Sequence-based Anomaly Detection Techniques. *arXiv preprint arXiv:2309.02854* (2023).

[29] Van-Hoang Le and Hongyu Zhang. 2021. Log-based anomaly detection without log parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 492–504.

[30] Van-Hoang Le and Hongyu Zhang. 2022. Log-based anomaly detection with deep learning: How far are we?. In *Proceedings of the 44th international conference on software engineering*. 1356–1367.

[31] Van-Hoang Le and Hongyu Zhang. 2023. Log parsing with prompt-based few-shot learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2438–2449.

[32] Zhijing Li, Qiuai Fu, Zhijun Huang, Jianbo Yu, Yiqian Li, Yuanhao Lai, and Yuchi Ma. 2024. Revisiting Log Parsing: The Present, the Future, and the Uncertainties. *IEEE Transactions on Reliability* (2024).

[33] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. 2007. Failure Prediction in IBM BlueGene/L Event Logs. 583 – 588. https://doi.org/10.1109/ICDM.2007.46

[34] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*. 102–111.

[35] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 eighth ieee international conference on data mining*. IEEE, 413–422.

[36] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. 2022. UniParser: A Unified Log Parser for Heterogeneous Log Data. In *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*. ACM, 1893–1901. https://doi.org/10.1145/3485447.3511993

[37] Jian-Guang Lou, Qiang Fu, Shenqi Yang, Ye Xu, and Jiang Li. 2010. Mining invariants from console logs for system problem detection. In *2010 USENIX Annual Technical Conference (USENIX ATC 10)*.

[38] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, Feifei Li, Changcheng Chen, and Dan Pei. 2020. Diagnosing Root Causes of Intermittent Slow Queries in Large-Scale Cloud Databases. *Proc. VLDB Endow.* 13, 8 (2020), 1176–1189. https://doi.org/10.14778/3389133.3389136

[39] Minghua Ma, Shenglin Zhang, Junjie Chen, Jim Xu, Haozhe Li, Yongliang Lin, Xiaohui Nie, Bo Zhou, Yong Wang, and Dan Pei. 2021. Jump-Starting Multivariate Time Series Anomaly Detection for Online Service Systems. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*. USENIX Association, 413–426.

[40] Minghua Ma, Shenglin Zhang, Dan Pei, Xin Huang, and Hongwei Dai. 2018. Robust and Rapid Adaption for Concept Drift in Software System Anomaly Detection. In *29th IEEE International Symposium on Software Reliability Engineering, ISSRE 2018, Memphis, TN, USA, October 15-18, 2018*. IEEE Computer Society, 13–24.

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

[42] Adam Oliner, Archana Ganapathi, and Wei Xu. 2012. Advances and challenges in log analysis. *Commun. ACM* 55, 2 (2012), 55–61.

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

[44] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. https://arxiv.org/abs/1908.10084

[45] Nils Reimers and Iryna Gurevych. 2020. Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. https://arxiv.org/abs/2004.09813

[46] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. 2009. Largescale system problem detection by mining console logs. *Proceedings of SOSP'09* (2009).

[47] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 117–132.

[48] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1448–1460.

[49] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306* (2018).

[50] Zhaoyang Yu, Minghua Ma, Chaoyun Zhang, Si Qin, Yu Kang, Chetan Bansal, Saravan Rajmohan, Yingnong Dang, Changhua Pei, Dan Pei, Qingwei Lin, and Dongmei Zhang. 2024. MonitorAssistant: Simplifying Cloud Service Monitoring via Large Language Models. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Porto de Galinhas, Brazil, July 15-19, 2024*. ACM, 38–49. https://doi.org/10.1145/3663529.3663826

[51] Chenyangguang Zhang, Tong Jia, Guopeng Shen, Pinyan Zhu, and Ying Li. 2024. MetaLog: Generalizable Cross-System Anomaly Detection from Logs with Meta-Learning. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.

[52] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 807–817.

[53] Nengwen Zhao, Honglin Wang, Zeyan Li, Xiao Peng, Gang Wang, Zhu Pan, Yong Wu, Zhen Feng, Xidao Wen, Wenchi Zhang, et al. 2021. An empirical investigation of practical log anomaly detection for online service systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1404–1415.

[54] Yue Zhao, Ryan Rossi, and Leman Akoglu. 2021. Automatic unsupervised outlier model selection. *Advances in Neural Information Processing Systems* 34 (2021), 4489–4502.

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

[56] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.