SLIM: a Scalable and Interpretable Light-weight Fault Localization Algorithm for Imbalanced Data in Microservice

ABSTRACT

In real-world microservice systems, the newly deployed service - one kind of change service, could lead to a new type of minority fault. Existing state-of-the-art (SOTA) methods for fault localization rarely consider the imbalanced fault classification in change service. This paper proposes a novel method that utilizes decision rule sets to deal with highly imbalanced data by optimizing the F1 score subject to cardinality constraints. The proposed method greedily generates the rule with maximal marginal gain and uses an efficient minorize-maximization (MM) approach to select rules iteratively, maximizing a non-monotone submodular lower bound. Compared with existing fault localization algorithms, our algorithm can adapt to the imbalanced fault scenario of change service, and provide interpretable fault causes which are easy to understand and verify. Our method can also be deployed in the online training setting, with only about 15% training overhead compared to the current SOTA methods. Empirical studies demonstrate the superior performance of our algorithm to existing fault localization algorithms in terms of both accuracy and model interpretability. We further show that our interpretable ruleset could construct the reusable knowledge base and achieve similar precision with expert experience.

KEYWORDS

microserivce fault localization, imbalanced classification, interpretability, submodular optimization, minorize-maximization

1 INTRODUCTION

Monolithic services have been progressively restructured into more refined modules, comprising of hundreds (or even thousands) of loosely-coupled microservices [12, 22, 23, 48]. Leading companies like Netflix, eBay and Alibaba have adopted this application model. Microservices offer several benefits that make them a powerful architecture, including simplification of application development, resource provisioning efficiency and flexibility. Despite these promising advantages, microservices introduce complex interactions among modular services, which can make on-demand resource provisioning challenge and potentially lead to performance degradation.

To enable engineers to resolve failure efficiently, fault localization is at the core of software maintenance for online service systems. With the advancement of monitoring and collecting tools, Metrics, Traces, and Logs have become the three fundamental elements of fault localization. Metrics refer to numeric data measurements taken at regular intervals of time, which help to understand the reasons behind the functioning of your application. Each trace records the process of a request being called through service instances and their operations [43, 46]. Logs provide detailed information on the system's running status and user behavior.

Though tremendous efforts have been devoted to software service maintenance and observability, in practice failures are inevitable due to the the increasing size and complexity of systems, which can result in significant economic losses and user dissatisfaction [12, 37, 44]. Analyzing the root causes of such performance issues is non-trivial and often error-prone, as hundreds of services may exhibit anomalies (e.g., network congestion and limited available cores) and propagate to dependent services. Moreover, large microservice systems are highly active and dynamic [12], with numerous service changes.

It is important to note that the faults in a service often occur within the change of service, e.g. initial period of deploying new service or code change of services. This is due to changes in system architecture, service version (with backward or forward compatibility), and insufficient testing of the new service itself [7, 34]. Google SRE book points out that about 70% of failures are caused by changes in services [11]. Additionally, the lack of adequate fault data for newly deployed services hinders the learning process, making root cause analysis algorithms hard to localize such faults, even leading to a chain of incidents. However, existing fault localization algorithms mainly focus on utilizing new deep-learning and machine-learning models to fully learn the information in the multi-source data (log, metric and trace), while ignoring the highly dynamic running-time status with numerous services in change and the limited training data. Thus, they are unable to quickly learn and respond to new faults caused by service changes with a small amount of fault data [15, 16].

Motivations. Existing fault localization methods for imbalanced classification generally rely on re-sampling the

training data [35, 36]. In fact, simply over-sampling the minority class samples or down-sampling the majority class samples may cause model overfitting and on the other hand cannot generate extra insights from the data. We show that those SOTA algorithms cannot achieve significant performance improvement through re-sampling in our experiment part 5.4.2. Besides, many deep learning-based fault localization methods [9, 21, 23, 41] which claim to have good fault localization performance, generally require several hours for a single training session. Thus, they fail to handle services experiencing frequent failures. In addition, those algorithms only offer simple binary classification results with a lack of interpretability, making it difficult for engineers to understand, diagnose, and further prevent faults in the next steps. Therefore, how to build an interpretable fault localization model that could quickly and accurately respond to newly deployed service fault patterns from a limited number of imbalanced failures is an appealing but challenging problem [17, 29].

We summarize three main technical challenges to build effective fault localization models for imbalanced datasets of newly deployed services as follows.

1) The first challenge arises from imbalanced data of service change (e.g., newly deployed service). We also note that simply applying re-sampling does not lead to performance improvement in our experiments.
2) The second challenge is the model interpretability for operating engineers. Good interpretability can help engineers to better understand the fault cause and possibly identify related risks. Unfortunately, most of the existing works on fault localization lacks interpretability.
3) The third challenge arises from the unbearable training overhead of existing models, primarily attributed to the dynamic runtime environment in the microservice scenario.

In this paper, we aim to design an interpretable classifier on highly imbalanced data via learning decision rule sets [19] for fault localization. It can be expressed in disjunctive normal forms (DNF, OR-of-ANDs), which enjoys good interpretability due to the logical clauses. An example of DNF models with two conditions is "IF (cpu_usage>80 AND file_disk_read>180) OR (file_disk_write>70 AND memory_usage >170) THEN $\hat{y} = 1$". It helps engineers to understand which key metrics are affected by the fault. As we only focus on building rules for the minority class (further called the positive class), thus it is an ideal interpretable model for imbalanced classification. Moreover, our model can achieve incremental (or online) training within minutes for every newly deployed services, which makes it deployable to a wide range of services with low cost.

In this paper, we propose a fault localization algorithm called **SLIM** (**S**calable and interpretable **L**ight-weight algorithm for **I**mbalanced data in **M**icroserverice) to address the aforementioned challenges. Here we summarize the main contributions as follows:

1) To the best of our knowledge, **SLIM** is the first fault localization algorithm to address the issue of imbalanced fault data in service change (e.g., newly deployed services) from an algorithm-level within the microservices environment.
2) Our fault localization algorithm can generate the interpretable rule set that could assist engineers in understanding the root causes of failures.
3) Our **SLIM** is efficient and can be deployed easily with a low cost. Compared to other algorithms, our model's training time is only around 15% of theirs in the most complex scenarios.
4) We apply our model's interpretable ruleset to two use cases, which replace human experts to build prior knowledge. The results show that our interpretable ruleset is comparable to expert knowledge and reduces 80% of the time required. Besides, our ruleset knowledge base beats the precision of other interpretable methods' knowledge base.
5) We have conducted extensive experiments on 359 failures from three systems including three open-source benchmark datasets [36]. The results show SLIM's effectiveness, efficiency and interpretability. We also apply our algorithm in the real running-time environment of the largest cloud service system provider in China. We give the detailed case study in the section of experiment. We also provide the demo available on the github [1].

2 BACKGROUND

2.1 Basic Concepts of Microservice System

In Figure 1, we show the basic concepts of the microservice system. A **Microservice System** is a highly efficient architecture comprising of **Microservices** that are loosely coupled and lightweight. **Microservices** are initiated by **instances**, which is the running version of this microservice on the **Physical Node**. The status of both the **Physical Nodes** and the **Instances** are described through **metrics** generated by monitoring tools such as Prometheus [8]. The **trace** describes the execution process of a request through service instances [3]. It has been supported by open-source solutions [5, 6]. The **Logs** are produced during each service invocation, which is generated by the logging statements and records the internal status of the invocation.

---

[1]https://anonymous.4open.science/r/SLIM-5B7F/
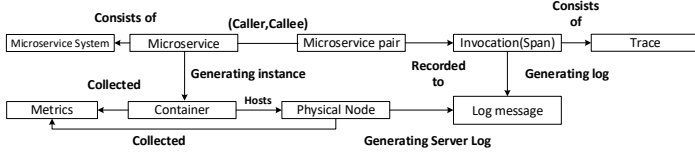
Figure 1: The Overview of Microservice System.

Table 1: Example of Monitoring Metrics of Enterprise Cloud Platform.

Entity Type	Example Metrics
Container	CPU utilization, memory utilization, File Disk I/O,network receive rate, transmit rate, network packet corrupt, network packet dropped
Virtual Machine	Similar with Container metrics
Database	Login number every Seconds, memory utilization,CPU utilization
Physical Host	Similar with Container metrics
Switch interface	Network rate, dropped packet, latency

2.2 The Monitorting of Enterprise Cloud Platform

Most enterprise operating teams use monitoring tools to observe the environment, including logs and performance metrics of hosts and applications. Microservice-based applications often utilize additional specialized monitoring, like Jaeger, which could track the RPC latency and errors between services' calls. We describe Alibaba Cloud Observability Suite, an application-aware observability platform, which is one of our sources of experimental data(dataset D) in this paper. This platform builds an observable data layer around Prometheus [8] service, Grafana [4] platform, and distributed tracing service [6], enabling metric storage analysis, link storage analysis, and heterogeneous data source integration. This platform could integrate multiple data sources. Example entity types and associated metric values are described in the table 1. The monitoring platform also could assist us in observing the services' calling status and the detailed log information of every applications or hosts.

2.3 Prior Work of Fault Localization

Several techniques employ the advanced data processing capabilities of machine learning and deep learning to achieve root cause analysis of multi-source data. MEPFL [54] incorporates multiple features, such as traces, configurations, and resource metrics. Three supervised machine learning models are trained using a fault injection-based training corpus to predict the root cause of microservices. Seer [23] captures

trace data at the RPC level and trains a hybrid deep learning network that combines a Convolutional Neural Network (CNN) with a Long Short-Term Memory (LSTM). The model's input and output neurons correspond to microservices, ordered in topological order from back-end microservices at the top to front-end microservices at the bottom. TraceAnomaly trys to vectorize every possible invocation paths and record their latencies. They judge the root cause by finding out the deepest service at every abnormal trace. Dejavu [36] leverages the GAT to learn the history fault and design the interpreter for their fault localization module.

2.4 Summary and Limitations of Prior Work

The existing deep-learning and machine-learning models are struggling to cope with such dynamic system states [38, 39, 49, 50]. The continually increasing services make it difficult to bear the training cost of the system. Many of these models also just output the simple binary classification results that are hard to be understood by operating engineers. For example, due to the limitation of using the number of service nodes that Seer itself employs as a model input, every time a new service is deployed, the network needs to be updated again. In its paper, it mentions that when diagnosing failures for a system with 100 microservices, Seer requires nearly 90 minutes for model updating.

2.5 The need of handling service change

Besides, due to the dynamic system environment, as shown in figure 3, the newly deployed service faces challenges in fault localization modeling due to a lack of observable data and historical failures. This makes it difficult for the fault localization model to adequately learn the failure patterns of the service and identify faults. In the part of 5.4.2, we analyze the impact of the frequency of faults of a specific service in the historical training data on the accuracy of the root cause analysis algorithm. We observe that as the frequency of failures of the service increased, the accuracy of the root cause analysis algorithm improved. When there is only one instance of a fault in the training set, the performance of the root cause analysis model is poor. To deal with this problem, we try to leverage the SMOTE [13] and its variants [31, 42] to create new re-samples by interpolating the minority class samples.

Unfortunately, the data imbalanced issue exists in two situations in the system. Firstly, the proportion of normal samples is much higher than that of failure samples, and this situation can be better solved by downsampling normal samples or oversampling fault samples. Secondly, the number of failure samples in the newly deployed service is much lower than that of failure samples in other services. This situation
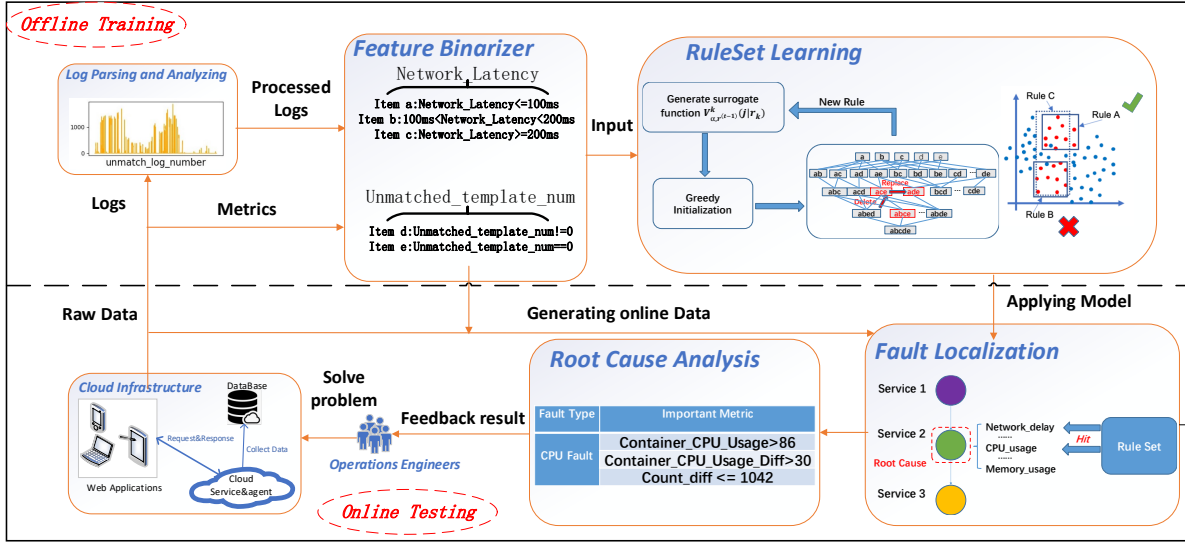
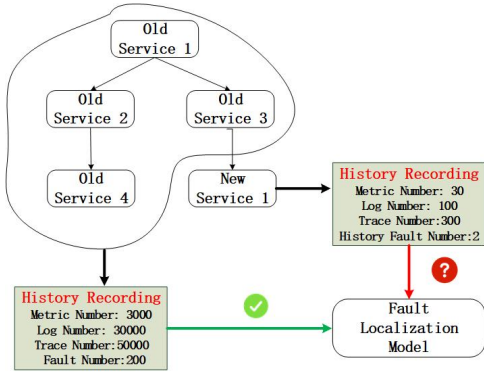**Figure 2: The Pipeline of SLIM.**

Figure 3: The Challenge of Service Change.

is difficult to solve by downsampling or oversampling because the number of failure samples in the new service is too rare, and re-sampling can lead to more noise and have a negative impact on the results instead. In the part of A, we combine the baseline models with these data re-sampling methods demonstrating that data re-sampling methods do not always work well.

To address these problems, we propose the SLIM, an **S**calable interpretable and **L**ightweight Fault Localization for **I**mbalanced Data **M**icroservice, which trains the rule set models to solve the fault localization with the imbalanced classification problems by optimizing the F1 score subject to cardinality constraints. Our model could incrementally learn new fault types from newly deployed services at a minute-level speed on a personal computer.

3 THE SLIM ALGORITHM

3.1 The Pipeline of SLIM

SLIM is an interpretable and scalable fault localization algorithm for microservices system. It identifies the (faulty) services that cause performance degradation in a microservice system, and provides explanations for operators to understand why. Figure 2 shows an overview of SLIM's pipeline, primarily involving 4 modules. Firstly, it processes logs and converts them into metrics as features. Then, the features are transformed into binary encoders. With these binary features as inputs, rules can be learned. Finally, the learned rules are used to vote out faults. We will now introduce these 4 modules in detail.

3.1.1 Log Extraction Module. This module extracts key log information and converts it to metric data. Operational log data is in unstructured format and not suitable for direct training. So we leverage log extraction methods [52] to deal with it. As Figure 4 shows, the procedure consists of log template parsing and analysis of template variation. We first parse the normal history log messages to construct a standard template base offline using drain [26]. These normal log templates serve as a historical reference to identify whether any new template exists in online log messages. Then we parse the online log messages and construct streaming time-series log templates. We compare the streaming time-series log templates with the template base and record the unmatched log template, along with their quantity, types, and other features. We aggregate these features by time interval, aligning them with the metrics' sample interval.

3.1.2 Feature Binarization Module. This module generates binarized features from the metrics obtained in the feature extraction module. We employ a bucketing strategy using a sequence of thresholds to cut the numerical features into discrete, binarized features. For example, in Figure 2, the network latency has a continuous distribution between 100-500ms. The data is discretized into multiple values, such as $Network\_Latency \leq 100$, $100 < Network\_Latency < 200$ and $Network\_Latency \geq 200$. For categorical features, we use one-hot encoding to generate binarized features. We apply this discretization process uniformly across the data distribution. These discrete values can be combined to form rules.

3.1.3 Efficient Rule Learning Method. This module digs out the rule set by our proposed novel classifier. The algorithm first identifies rule sets for each fault type. Namely, for each fault type we train a sub-model and obtain a rule set. Then, those which hit the rules in the rule set are classified as corresponding fault samples. The detailed rule learning method will be discussed in Section 3.2.

3.1.4 Fault Localization. This module provides the final results, including the localization of fault types and the localization of fault services. We design fault ranking methods by adopting a voting mechanism that takes into account both the hit counts of each rule set and the rule's confidence.

3.2 Detailed Procedure of Rule Set Learning

In the section, we introduce our rule learning methods, including Rule Set Selection in Section 3.2.2 and Efficient Rule Generation in Section 3.2.3. Before diving into the details, we first predefine some key notations.

3.2.1 Notations and Preliminaries. Given a dataset $X = \{(x_i, y_i)\}_{i=1}^{n}$, $x_i \in \{0, 1\}^d$ is the binary feature vector obtained from feature binarization and $y_i \in \{0, 1\}$ is the true label indicating the belongingness of a given fault type. Here, d is the size of the feature index set Γ. A sample (x_i, y_i) is positive if $y_i = 1(abnormal)$ and negative if $y_i = 0(normal)$. We call the i-th sample is covered by feature j if $x_{i,j} = 1$. For example, given $j \in d$, the j-th feature $Network\_delay > 200ms$, $x_{i,j} = 1$ means for the i-th sample, $Network\_delay > 200ms$ and $x_{i,j} = 0$ means $Network\_delay \leq 200ms$. Denote $h_i \in \{0, 1\}$ as the prediction of the i-th sample. For a certain fault type, we aim to predict y from the dataset X using an interpretable rule set.

Rule and rule set. Our classifier is a rule set that consists of rules. We now give the definitions of these two ingredients and the related notations.

DEFINITION 1 (**RULE**). *A rule r is a set of feature indices, i.e. r is a subset of Γ. A sample (x_i, y_i) is covered by a rule r if and only if $r \subseteq \{j \in \Gamma | x_{i,j} = 1\}$.*
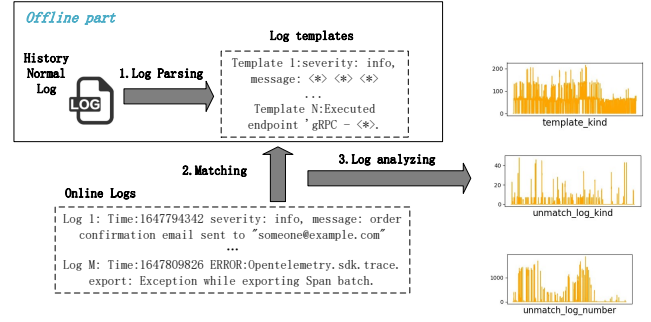
Figure 4: Log Extraction Module: Log Parsing, Matching and Analyzing.

DEFINITION 2 (**RULE SET**). *A rule set s consists of multiple rules, and serves as a classifier, which classifies a sample as positive if the sample is covered by at least one rule in s, and as negative if there is no rule in s that covers it. Therefore, the predicted label h_i can be calculated via $\bigvee_{r \in s}(\bigwedge_{j \in r} x_{i,j})$.*

Let X_j, X_r and X_s denote the set of samples covered by the j-th feature, the rule r and the rule set s, respectively. In the following we use these different subscripts to distinguish sets of samples covered by different features/rules/sets, i.e. $X_j = \{i | x_{i,j} = 1\}$, $X_r = \{i|(\bigwedge_{j \in r} x_{i,j}) = 1\}$, $X_s = \{i| \bigvee_{r \in s}(\bigwedge_{j \in r} x_{i,j}) = 1\}$. According to the relationships among the features, rules and rule sets, we get $X_r = \bigcap_{j \in r} X_j$ and $X_s = \bigcup_{r \in s} X_r$. We define a set operator $^+$ as a positive sample filter, meaning that X'^+ returns a set containing all positive samples in the arbitrary given set X'.

Problem Formulation. The common evaluation metrics, accuracy and error rate, are no longer applicable in imbalanced classification as they are prone to be dominated by the majority class [30]. To address this issue, we choose **F1 score** instead, which combines both precision and recall and cares about the performance of the minor positive samples:

$$F1(s) = \frac{2 \sum_i y_i h_i}{\sum_i h_i + \sum_i y_i}. \tag{1}$$

Since $\sum_i y_i h_i$ is the number of correctly classified positive samples, $\sum_i h_i$ is the number of positively predicted samples, $\sum_i h_i$, $\sum_i y_i h_i$ and $\sum_i y_i$ can be rewritten as $|X_s|$, $|X_s^+|$ and $|X^+|$, respectively. Formally, we formulate the F1 score as follows:

$$F1(s) = \frac{2|X_s^+|}{|X_s| + |X^+|} = \frac{2| \cup_{r \in s} X_r^+|}{| \cup_{r \in s} X_r| + |X^+|}. \tag{2}$$

To maximize F1 score, we have the following optimization problem:

$$\max_{s \subseteq \Omega} \frac{2| \cup_{r \in s} X_r^+|}{| \cup_{r \in s} X_r| + |X^+|}, \tag{3}$$
$$\text{s.t. } |s| \leq K, \ \Omega = \{r | |r| \leq l, r \subseteq \Gamma\}.$$

where we set a predefined parameter l to be the maximum feature length of a rule, and K to be the maximum number of rules in a rule set. These constraints are to ensure the interpretablity of rules. By taking the logarithm, we can rewrite the objective in the following form:

$$\max_{s \subseteq \Omega} \ \log(|\cup_{r \in s} \mathcal{X}_r^+|) - \log(|\cup_{r \in s} \mathcal{X}_r| + |\mathcal{X}^+|). \quad (4)$$

3.2.2 Rule Set Selection. We now present the details of our efficient rule selection method. Let us rewrite the two logarithm components of (4) as $G(s) \triangleq \log(|\cup_{r \in s} \mathcal{X}_r^+|)$ and $C(s) \triangleq \log(|\cup_{r \in s} \mathcal{X}_r| + |\mathcal{X}^+|)$. As logarithm function is non-decreasing and concave, both $G(s)$ and $C(s)$ are non-negative monotone submodular functions [10]. Consequently, the objective function can be viewed as a difference between two submodular functions. Our proposed method, which is referred as SLIM, is based on the method *DistortedGreedy* [25], for maximizing the difference between a non-negative monotone submodular function and a modular function. We will show that by introducing the notation curvature, *DistortedGreedy* is applicable to our problem. We first define the curvature γ of $C(s)$

$$\gamma \triangleq 1 - \min_{r \in \Omega} \frac{C(r|\Omega \setminus \{r\})}{C(r|\emptyset)} \quad (5)$$

to measure the closeness of $C(s)$ to a modular function, and γ is unknown a priori. The complete procedure for rule set selection is summarized in Algorithm 1. Given the training dataset, the maximal number of rules and the limitation on the length of rules, and by initializing the rule set as an empty set, we iteratively add a rule r^* maximize a distorted marginal gain of r with a parameter α, i.e.,

$$\max_{r} \ \alpha G(r|s) - C(r|s), \quad (6)$$

where $G(r|s) \triangleq G(s \cup \{r\}) - G(s)$ and $C(r|s) \triangleq C(s \cup \{r\}) - C(s)$ denote the marginal gains of G and C when adding r to s, which is stated in line 6 of Algorithm 1.

Algorithm 1: Rule Set for Imbalanced Data Set

---

1 **Input :** Training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, cardinality K, curvature γ, maximal size of a rule l.

2 **Output :** Rule set s.

3 Let $s_0 \leftarrow \emptyset$;

4 **for** $i = 0, 1, ..., K-1$ **do**

5 $\quad \alpha_i \leftarrow (1 - \frac{\gamma}{K})^{K-(i+1)}$;

6 $\quad r^* \leftarrow \arg\max_r \ \alpha_i G(r|s) - C(r|s)$;

7 $\quad$ **if** $\alpha_i G(r^*|s) - C(r^*|s) > 0$ **then**

8 $\quad\quad$ | $\quad s_{i+1} \leftarrow s_i \cup \{r^*\}$;

9 $\quad$ **end**

10 **end**

11 **Return** s_{K-1}

Similar to *DistortedGreedy*, we adaptively update the trade-off between $G(r|s)$ and $C(r|s)$ using α in line 5 of Algorithm 1. In early stages, a small value of α is adopted to select rules with higher *precision*. The value of α is gradually increased to improve the *recall* of the rule set. In other words, SLIM tends to select the rules with higher *precision* in the early iteration steps and focuses on the rules with higher *recall* later. The rationale behind the α updating strategy is that when first focusing on the rules with high *precision* and low *recall*, SLIM can achieve higher precision and later improve the recall by including more rules. However, if the rules with high *recall* but low *precision* are given more priority in early stages, it is difficult to eliminate the effect of the false positive samples.

To better illustrate this, we show a toy example in Figure 5. Given a dataset with 20 positive and 100 negative samples, and our goal is to select 2 rules from 3 candidate rules, namely rules A, B and C, where rule A covers 10 positive and 1 negative samples, rule B covers the rest 10 positive samples which are not covered by rule A and 1 additional negative sample, and rule C covers 18 positive samples and 5 negative samples. We first discuss the scenario that we replace the α update strategy in line 3 of Algorithm 1 with $\alpha_i = 1$. At the first iteration for rule A, $|\mathcal{X}^+| = 20$, $|\cup_{r \in s} \mathcal{X}_r| = 11$, $|\cup_{r \in s} \mathcal{X}_r^+| = 10$, then the marginal gain of rule A is $\log(10/(20+11)) \approx \log(0.31)$. Similarly, the marginal gains of rule B and rule C are given as $\log(10/(20+11)) \approx \log(0.31)$, $\log(18/(20+18+5)) \approx \log(0.42)$, respectively. In this scenario, SLIM will select rule C in the first iteration and rule B(or A) in the second iteration. Finally, SLIM constructs a rule set which covers 20 positive samples and 6 negative samples. However, with the proposed α update strategy in line 3 of Algorithm 1, at the first iteration (corresponding to $K = 2$, $i = 0$, $\gamma = 1$, and $\alpha = 0.5$), the marginal gains of rule A, B and C are given as $0.5 \times \log(10) - \log(31)(\approx \log(0.102))$, $0.5 \times \log(10) - \log(31)(\approx \log(0.102))$ and $0.5 \times \log(18) - \log(43)(\approx \log(0.099))$, respectively. Then SLIM will return a better rule set that consists of rule A and rule B, which covers only 2 negative samples. We also give the theoretical guarantee for the proposed method at appendix B

3.2.3 Efficient Rule Generation. Algorithm 1 involves finding a rule r to maximize a distorted marginal gain of r, i.e. solving problem (6). As the number of possible rules is exponential with number of features, solving problem (6) is NP-hard. To address this issue, we propose an efficient rule generation method, which solves (6) approximately. Because s is independent of r, (6) can be reduced to

$$\max_{|r| \leq l} \ \alpha \log(|\mathcal{X}_r^+ \cup \mathcal{X}_s^+|) - \log(|\mathcal{X}_r \cup \mathcal{X}_s| + |\mathcal{X}^+|). \quad (7)$$

Notice that a rule r is a set of feature indices, so finding an optimal rule is equivalent to finding a set of features. The
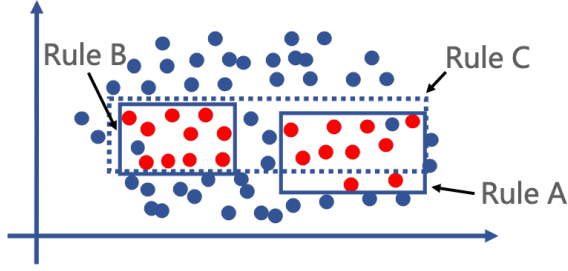
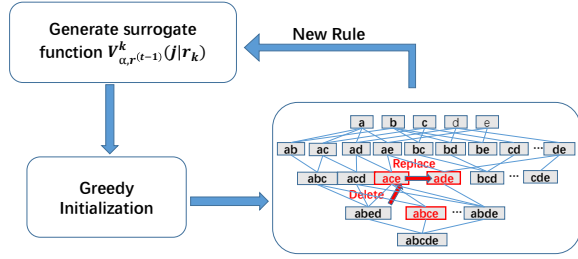**Figure 5: Example of the proposed rule selection strategy.**

Figure 6: Framework of Rule Generation.

expression in (17) can be further rewritten as:

$$W(r) \triangleq \alpha \log(f(r)) - \log(g(r)) \tag{8}$$

where $f(r) \triangleq |(\cap_{j \in r} \mathcal{X}_j)^+ \cup \mathcal{X}_s|$ and $g(r) \triangleq |(\cap_{j \in r} \mathcal{X}_j) \cup \mathcal{X}_s| + |\mathcal{X}^+|$.

Directly maximizing $W(r)$ is difficult. Although $f(r)$ is a supermodular function, the presence of logarithm function makes the property of $\log(f(r))$ non-trivial. In our algorithm, $W(r)$ is maximized using MM algorithm [27], which iteratively increases the value of the objective function by maximizing a tight lower bound. We propose a proper lower bound of $W(r)$ by finding a lower bound of $\log(f(r))$ and an upper bound of $\log(g(r))$ separately.

Motivated by the modular upper bounds of submodular functions presented in [28, 47], two proper lower bounds of $f(r)$ for all $r \subseteq \Gamma$ are given as

$$L^1_{f,r^{(t)}}(r) \triangleq f(r^{(t)}) - \sum_{j \in Q_1} f(j|r^{(t)} \setminus \{j\}) + \sum_{j \in Q_2} f(j|\emptyset) \le f(r)$$

$$L^2_{f,r^{(t)}}(r) \triangleq f(r^{(t)}) - \sum_{j \in Q_1} f(j|\Gamma \setminus \{j\}) + \sum_{j \in Q_2} f(j|r^{(t)}) \le f(r)$$

where $r^{(t)}$ denotes the current estimation of r, $Q_1 = r^{(t)} \setminus r$ and $Q_2 = r \setminus r^{(t)}$. These two inequalities hold for all possible $r^{(t)}$, and the equality is achieved when $r = r^{(t)}$. We find an upper bound of $\log(g(r))$ by utilizing the concavity of logarithm functions. As $\log(x) \le \log(x_0) + \frac{1}{x_0}(x - x_0)$, then a tight upper bound of $\log(g(r))$ is readily given as, which

holds for any $r^{(t)}$,

$$\log(g(r)) \le \log(g(r^{(t)})) + \frac{1}{g(r^{(t)})}(g(r) - g(r^{(t)})). \tag{9}$$

Combining the bounds obtained above, we derive two tight lower bounds of $W(r)$ for $|r| \ge 1$ as follows,

$$W(r) \ge \alpha \log(L^1_{f,r^{(t)}}(r)) - \frac{g(r)}{g(r^{(t)})} = V^1_{\alpha,r^{(t)}}(r), \tag{10}$$

$$W(r) \ge \alpha \log(L^2_{f,r^{(t)}}(r)) - \frac{g(r)}{g(r^{(t)})} = V^2_{\alpha,r^{(t)}}(r). \tag{11}$$

The problem of maximizing $W(r)$ is translated to maximizing $V^1_\alpha(r|r^{(t)})$ and $V^2_\alpha(r|r^{(t)})$. To step further, we give the detailed procedure of translation at the appendix A.

Maximizing a non-monotone submodular function subject to cardinality constraints has been extensively studied in the literature. Specifically, SLIM maximizes $V^1_\alpha(r|r^{(t)})$ and $V^2_\alpha(r|r^{(t)})$ by using a simple local search method. As shown in [33], by identifying the cardinality constraint as a matroid constraint, the local search method can provide at least 1/4-approximation to the optimum.

LEMMA 1. *Both $V^1_\alpha(r|r^{(t)})$ and $V^2_\alpha(r|r^{(t)})$ are non-monotone submodular functions.*

Maximizing a non-monotone submodular function subject to cardinality constraints has been extensively studied in the literature. Specifically, SLIM maximizes $V^1_\alpha(r|r^{(t)})$ and $V^2_\alpha(r|r^{(t)})$ by using a simple local search method. As shown in [33], by identifying the cardinality constraint as a matroid constraint, the local search method can provide at least 1/4-approximation to the optimum.

Fig.6 shows the overall framework of our rule generation method. At the tth iteration, we first generate surrogate functions of $W(r)$, i.e. $V^1_{\alpha,r^{(t)}}(r)$ and $V^2_{\alpha,r^{(t)}}(r)$, according to current estimation $r^{(t)}$. Then we maximize $V^1_{\alpha,r^{(t)}}(r)$ and $V^2_{\alpha,r^{(t)}}(r)$ utilizing local search technique and arrive at a new estimation $r^{(t+1)}$. Our method only involves the set operation and is hence a computational efficient method. The computational complexity of our method can be further improved by permitting early stopping, i.e, terminating the local search if no significant improvement is achieved by replacing features.

Algorithm 2 summarizes our MM-based method for generating an optimal rule by pseudocode. The lines 4-7 present the initialization of each MM iteration, where we insert l features to an empty set greedily. Local search is then performed to refine the solution, as stated in lines 8-15. The replacement operation, which replaces a feature in r with another feature in Γ if the objective function can be improved, is performed in the local search procedure only. To perform the deletion operation, we extend Γ by including the empty set, as shown

Algorithm 2: GenerateRule

---

1 **Input :** l: Maximal size of a rule. α: trade-off parameter.
2 **Output :** Rule r.
3 **Initialize** r_0 *by GreedRatio*;
4 **for** $t = 1, 2, ...$ **do**
5     $r_k \leftarrow \emptyset, k \in \{1, 2\}$;
6   **for** $i = 1, 2, ..., l$ **do**
7        $j^* \leftarrow \arg\max_{j \in \Gamma} V^k_{\alpha, r^{(t-1)}}(j|r_k)$;
8        $r_k \leftarrow r_k \cup \{j^*\}$;
9   **end**
10   **repeat**
11        $\tilde{r} \leftarrow r_k$;
12   **for** $i \in r_k$ **do**
13   **Define** $V(j) = V_{\alpha, r^{(t-1)}}(j|r_k \setminus \{i\})$;
14           $j^* \leftarrow \arg\max_{j \in \Gamma \cup \{\emptyset\}} V(j)$;
15   **if** $V(j^*) > 0$ **then** $r_k \leftarrow (r_k \setminus \{i\}) \cup \{j^*\}$;
16   **end**
17   **until** $\tilde{r} = r_k$;
18     $r^{(t)} \leftarrow \arg\max_{r \in \{r_1, r_2\}} W(r)$;
19   **if** $r^{(t)} = r^{(t-1)}$ **then Return** $r^{(t)}$;
20 **end**

---

in line 12 of Algorithm 2, so that a feature in r can be deleted by replacing it with the empty set. To conclude, we have presented an efficient rule generating method, which iteratively improves the objective and is guaranteed to converge to a local optimum of the objective function.

3.3 Detailed procedure of Fault Localization

In this section, we introduce the detailed procedure of fault localization. For the localization of fault type, let the R^* denote the selected fault type result, R the collection of all fault types' rule sets, x_i the i-th sample in the current time period, r_j the j-th fault type and X all samples. $H_{r_j}(x_i)$ is an indicator function that takes the value 1 if x_i is hit by the rule set r_j and 0 otherwise. If the i-th sample is hit by multiple rules in the j-th ruleset, we select the rule with the highest precision $Precision^{max}_{r_j}$ at the training set to hit the sample. So we calculate the probability of the i-th sample belonging to the j-th fault type as Equation (12) writes. Then in Equation (13) we sum up the sample-level probability for each fault-type rule set j and pick out the fault-type with the highest value as the root cause:

$$P(r_j|x_i) = Precision^{max}_{r_j} * H_{r_j}(x_i), \qquad (12)$$

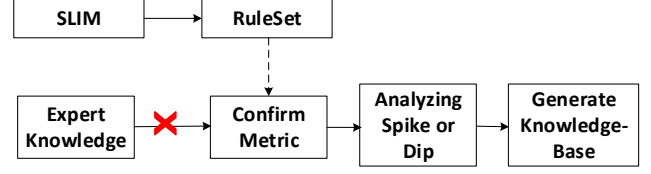$$R^* = \arg\max_{r_j \in R} \sum_{x_i \in X} P(r_j|x_i). \qquad (13)$$

Figure 7: The overview of Knowledge Base generation

For the localization of service, let the S^* denote the selected service result and X_k all samples in the k-th service. $x_m \in X_k$ is the m-th sample in the k-th service and $H_{r_j}(x_m)$ means whether x_m is hit by ruleset r_j. We first group the samples by service. As Equation (14) demonstrates, similar to fault type localization, the probability for each sample is the highest precision's rule in r_j when these rules in r_j hits the sample and otherwise 0. Then, we individually sum up the product of all samples hit by the rules in each service multiplied by the corresponding rule precision score according to Equation (15). We sort the probability results of every service and choose the service with the highest value as the root cause.

$$P(r_j|X_k) = \sum_{x_m \in X_k} Precision^{max}_{r_j} * H_{r_j}(x_m), \qquad (14)$$

$$S^* = \arg\max_{X_k \in X} \sum_{r_j \in R} P(r_j|X_k). \qquad (15)$$

The fault localization module could help us to confirm the failure service and fault type of the failure. We give a detailed evaluation of the effect of our model in the experiment part.

4 THE APPLICATION OF INTERPRETABLE RULESET

Our interpretable ruleset could assist engineers to confirm the root cause and find out the most relevant metric for the anomaly. It makes the model could corporate with expert knowledge for troubleshooting and debugging system errors. On this basis, we go a step further to apply our model to generate the knowledge base for existing fault localization algorithms. We will introduce these applications in the following part.

4.1 Overview of the Knowledge Base Generator

Usually, an anomaly knowledge base is constructed manually to store expert experience and quickly localize history fault types. When an anomaly firstly occurs, experts analyze its key features and generate fault fingerprints for the Knowledge Base. The fingerprint will help engineers to fast solve the problem again in the future.

Therefore, expert knowledge plays an important role for many algorithms [20, 51]. These algorithms leverage expert

Table 2: Benchmark dataset statistics.

Dataset	Features	Samples	Service	faults classes
$\mathcal{A}$	124	62249	17	18
$\mathcal{B}$	501	412714	29	20
$\mathcal{C}$	8	469988	43	2
$\mathcal{D}$	12000	7200	2173	5

knowledge to describe every fault type and construct the prior knowledge from metrics and trace data. This is called the knowledge base or case base. Although the knowledge base is necessary for real systems, it is also expensive for the company to recruit experts and build a knowledge base manually. Thus, our interpretable ruleset can help to construct the knowledge base and reduce costs. Fig. 7 shows an overview of the knowledge base generation. We try to replace the expert knowledge component in these algorithms with our rule set in order to build the knowledge base required for diagnosing faults in their models.

4.2 CloudRCA

CloudRCA [51] leverages RobustSTL to extract the abnormal metrics and identify important system metrics using the expert knowledge base. The selected metrics are learnt via a Knowledge-informed Hierarchical Bayesian Network (KHBN) to perform root cause analysis.

In our implementation, we replace the RobustSTL and expert knowledge with our interpreter to find out the most important metric sequence. We first pass labeled training data through our SLIM, where the model analyzes the fault cases to give the key ruleset. Then, we construct the feature matrix according to the ruleset including the key metric and log information. Finally, the KHBN completes the root cause analysis. We finish the experiment with the A dataset and evaluate the CloudRCA's performance.

4.3 MicroCBR

Similarly, we also integrate our ruleset into MicroCBR [20], which leverage the labeled anomaly case to construct the knowledge base and perform root cause analysis through case-based reasoning. Each case records a specific root cause and its solution, along with a set of anomalies detected from resource metrics, logs and other operating information. The precision of case-based reasoning is always dominated by these abnormal metrics.

We leverage our ruleset to automatically construct anomaly knowledge for its knowledge-base. First, the SLIM to learn the labeled data. Then, our ruleset will show the important metrics for the all fault type. MicroCBR aims to record the time-series metrics' fingerprint, which includes every

key metric's spike or dip. We leverage our ruleset to assist the MicroCBR to select the key metrics and construct fingerprints.

5 EXPERIMENTS

In this section, we perform experiments on benchmark datasets to show the performance of SLIM in comparison with the state-of-the-art fault localization algorithms.

5.1 Experimental Setup

5.1.1 Datasets. We conduct experiments on three public datasets, which are denoted as dataset $\mathcal{A}$, $\mathcal{B}$ [1], C [2], and $\mathcal{D}$, respectively.

Dataset $\mathcal{A}$ and $\mathcal{B}$ are obtained from two different production service systems, which are both injected in 18 types of faults that can be summarized as: (*i*) CPU exhaustion on containers, physical servers and middleware. (*ii*) packet loss, delay on service and physical node. (*iii*) database connection limit and close(just for dataset $\mathcal{A}$); (*iv*) low free memory at JVM/Tomcat (just for dataset $\mathcal{B}$); (*v*) Disk I/O exhaustion (just for dataset $\mathcal{B}$) [36]. The dataset C is generated by the train ticket booking microservice system [35, 53], where the fault classes include the network delay and CPU consuming. The dataset $\mathcal{D}$ comes from the real-world system that is one of the biggest cloud services provider(we refer to it as **Company ALC** for brevity). It consist of 35 incidents occurred in our cloud platform. These incidents are collected and verified which services are the root cause by our SRE team. For every fault, we have about 12000 metrics, collected during 3600 seconds(half hour before and half hour after the anomaly was reported) from 2173 microservices. We summerize the number of features, number of samples, number of services, and number of faults classes in Table 2.

5.1.2 Baseline Algorithms. We compare our proposed SLIM with several state-of-the-art fault localization algorithms, including five supervised methods, i.e. Dejavu, Seer, MEPFL-RandomForest(MEPFL-RF), Multilayer Perceptron(MEPFL-MLP), decision tree, Eadro, Murphy, Sage and AutoMap. Dejavu is an actionable and interpretable fault localization method for recurring failures, where graph attention networks is used to localize the fault [36]. Murphy [24] based on a Markov Random Field (MRF) that can take advantage of such loose associations to reason about how entities affect each other in the context of a specific incident. Sage [21] based on the Conditonal VAE that could simulate the service's status and counterfactual the system by restore the service's abnormal metric and confirm the root cause. Eadro [32] is similar with Dejavu that leverages the Graph Attention Networks to learn the log, metric and trace. They try to embedding log into the node features by Seer [23] captures the RPC-level graph dependency and metric by training a

Table 3: Accuracy comparison of different root cause localization algorithms.

Dataset	Algorithm	Category	A@1	A@1⇑	A@2	A@2⇑	A@3	A@3⇑	Kappa Analysis
	SLIM		0.791	–	0.837	–	0.86	–	0.7649
	DecisionTree		0.532	48.7%	0.635	31.8%	0.656	31.1%	0.5019
	Seer		0.482	64.1%	0.594	40.9%	0.643	33.7%	0.4507
$\mathcal{A}$	MEPFL(RF)	Supervised	0.698	13.3%	0.837	0%	0.86	0%	0.7047
	MEFPL(MLP)		0.452	75%	0.574	45.8%	0.603	42.6%	0.3281
	Dejavu		0.771	2.6%	0.903	−7.3%	0.934	−7.9%	0.7604
	Eadro		0.741	6.7%	0.86	−2.7%	0.903	−4.8%	0.7214
	AutoMap	Unsupervied	0.336	135%	0.435	92.4%	0.489	75.6%	0.2745
	Sage	Semi-supervised	0.635	2.6%	0.771	8.6%	0.837	2.7%	0.6079
	Murphy		0.656	2.6%	0.791	5.8%	0.837	2.7%	0.6242
	SLIM		0.673	–	0.75	–	0.827	–	0.6423
	DecisionTree		0.559	20.4%	0.603	24.4%	0.635	30.2%	0.5331
	Seer		0.503	33.8%	0.564	33%	0.603	37.1%	0.4882
$\mathcal{B}$	MEPFL(RF)	Supervised	0.603	11.6%	0.635	5.1%	0.756	9.4%	0.5803
	MEFPL(MLP)		0.487	38.2%	0.513	46.2%	0.603	37.1%	0.4358
	Dejavu		0.662	1.7%	0.712	5.3%	0.756	9.4%	0.6342
	Eadro		0.635	6.0%	0.698	7.5%	0.788	4.5%	0.6079
	AutoMap	Unsupervied	0.258	161%	0.342	119%	0.379	118%	0.2132
	Sage	Semi-supervised	0.513	31.2%	0.635	18.1%	0.712	16.2%	0.4570
	Murphy		0.564	19.3%	0.662	13.3%	0.756	9.4%	0.5185
	SLIM		0.931	–	0.967	–	0.992	–	0.9132
	DecisionTree		0.771	20.8%	0.86	12.4%	0.90	10.2%	0.7332
	Seer		0.82	13.5%	0.843	14.7%	0.882	12.5%	0.7913
C	MEPFL(RF)	Supervised	0.89	4.5%	0.956	1.2%	0.967	2.6%	0.8607
	MEFPL(MLP)		0.91	2.3%	0.967	0%	0.985	0.7%	0.8764
	Dejavu		0.92	0.4%	0.956	1.1%	0.992	0%	0.8832
	Eadro		0.90	3.3%	0.956	1.1%	0.992	0%	0.8642
	AutoMap	Unsupervied	0.534	74.3%	0.624	55%	0.741	33.9%	0.4213
	Sage	Semi-supervised	0.82	13.5%	0.86	8.2%	0.90	10.2%	0.7862
	Murphy		0.843	10.4%	0.86	12.4%	0.90	10.2%	0.8135

hybrid deep learning network that combines a CNN (Convolutional Neural Network) with an LSTM (Long Short-Term Memory). MEPFL-RandomForest(MEPFL-RF) and Multilayer Perceptron(MEPFL-MLP) [54] treat fault localization as a classification problem and solve it using traditional machine learning methods RandomForest and Multilayer Perceptron, respectively. We also compare our method with decision tree due to its interpretablity. AutoMap leverages the multi-dimension metrics to dynamically generate service relationship graph, and then leverages the random walk algorithm to localize the fault from the graph [40].

5.1.3 Experiment Environment and Parameter Tuning. We implement SLIM using Python 3.7 and Go language. All the experiments are conducted on a personal computer with 3070ti, 32GB RAM and 5800X processors with 6 cores. We tune parameters for all methods by 5-fold cross-validation.

Specifically, for our model SLIM, we choose the number of rules, i.e. parameter K, from $\{2, 4, 8, 12\}$, and limit the length of a rule to no more than 6 to ensure interpretability, i.e., $l = 6$. At the table 4, we show the detail analysis of Feature Binarizer module. We test the performance impact of our rule set algorithm under different interval partition quantities and ultimately determined that the default optimal value is **100** for the partition quantity of Feature Binarizer. For MEPFL-MLP and Seer, we adjust the number of neurons from [20,30,40] and the learning rate from [1e-4,5e-5,1e-5]. For DecisionTree and MEPFL(RF), the number of samples at each leaf node is tuned from 1 to 100 and the number of trees is tuned in $\{1000, 2000, 3000\}$. For Dejavu, we set the parameters according to the suggestion in [36].

Table 4: Accuracy comparison of different NumThresh

Dataset	Bining Number	A@1	A@2	A@3	A@4	A@5
$\mathcal{A}$	50	0.791	0.814	0.837	0.860	0.860
	75	0.791	0.837	0.837	0.860	0.860
	100	0.791	0.837	0.860	0.884	0.884
	125	0.744	0.791	0.837	0.837	0.837
	150	0.791	0.837	0.837	0.837	0.837
$\mathcal{B}$	50	0.632	0.673	0.712	0.788	0.884
	75	0.632	0.673	0.788	0.827	0.884
	100	0.673	0.75	0.827	0.884	0.967
	125	0.673	0.712	0.788	0.827	0.827
	150	0.673	0.712	0.827	0.884	0.884
C	50	0.82	0.86	0.891	0.967	1
	75	0.82	0.86	0.891	1	1
	100	0.931	0.967	0.992	1	1
	125	0.931	0.967	0.967	1	1
	150	0.891	0.967	0.992	1	1

5.2 Performance on Fault Localization

5.2.1 Evaluation Metrics. **Top-k Accuracy**, which is referred as **A@k**, is used to measurement the performeance of each methods. Top-k Accuracy computes the probability that the root causes can be located within the top k service instances among all candidates. Higher A@k indicates more accurate of the root cause localization. Here we measure the performance of each method using $A@1$, $A@2$, and $A@3$.

Kappa Analysis, which is Cohen-Kappa analysis [45], a statistical method used to measure the inter-rater reliability. It is generally thought to be a more robust measure than a simple percent agreement calculation. Due to the requirement for precision data in Kappa analysis, and considering that our model provides root cause rankings rather than precision, we select the top-1(A@1) result from the ranking as the final localization outcome to carry out the Kappa test.

5.2.2 Performance. We present the fault localization results in table 3, where the result are averaged over 5 independent trials. From table 3, we see that the proposed SLIM achieves highest Top-1 accuracy and Cohen-Kappa value on dataset $\mathcal{A}$,$\mathcal{B}$ and C. This due to we employ F1 score as objective function, which is robust to data imbalance. In contrast, Dejavu simply resamples the data to balance the number of data in each class, thus performs slightly worse than SLIM [36]. Eadro and Dejavu share similar performance outcomes because they employ the same methodology. Sage and murphy leverage the counterfactual method to restore the system and localize the root cause. Due to the Sage and Murphy is semi-supervised counterfactual inference methods, their performance experiences a slight decrease compared to supervised algorithms. However, they are more suitable

for fault recovery and exploration in service change. Seer, Decision Tree, MEPFL-RF, and MEPFL-MLP do not have specific designs to address imbalanced data, leading to inferior performance.

AutoMap performs poorly on three datasets. This is because these approaches do not make use of any historical faults information until the ground truth of similar historical faults are identified. Some critical intermediate steps in these methods, such as anomaly detection and similarity evaluation, despite being carefully designed, are entirely unsupervised. As a result, they may be susceptible to confusion from irrelevant abnormal changes in other metrics, which can be caused by noise or fluctuations, particularly when the number of metrics or fault units is high. On the contrary, SLIM focuses on the key metrics from the rule set that is generated by historical failures.

5.2.3 Overhead. In Figure 8, we evaluate the training overhead for each algorithms on three datasets. From Figure 8, we can see that our approach has lower training costs compared to all deep learning and some of machine learning models. As the deep learning based methods require more training time, thus the computational costs of Seer and Dejavu are much higher than that of the rest methods. We note that as Seer and Dejavu are highly rely on the current service topology diagram, as the models need to be retained once the service topology is change (such as there is a new service deployed). Overall, the high training overhead makes Seer and Dejavu unsuitable for scenarios that needs the fault localization methods to be adapt quickly.

5.3 Evaluation of Real-World System(Dataset D)

We present the fault localization results in table 5. Due to the large number of real system services and the high requirements for algorithmic overhead, the comparison algorithms we previously used, such as Dejavu and Seer, have excessive computational cost for deep learning models and cannot adapt well to the system requirements. Therefore, we only compare methods like Decision Tree and RandomForest. As shown in the table, our model still maintains better performance than other methods. In addition, we also compared the algorithm Ripper [18], which is another rule-based algorithm, but it does not optimize for imbalanced data. Compared with Ripper, we made for the imbalanced dataset, our model has fewer false positives, resulting in a more accurate ranking of fault.
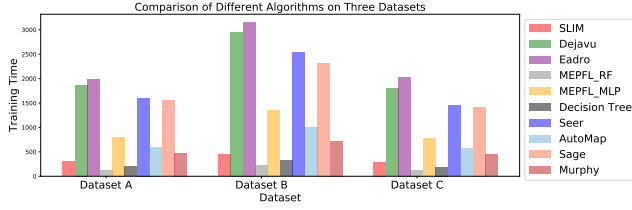
Figure 8: The Overhead of all Algorithms on Benchmark Datasets.

Table 5: Comparison of different root cause localization algorithms at Real-World System.

| Dataset	Algorithm	A@1	A@1⇑	A@2	A@2⇑	A@3	A@3⇑
$\mathcal{D}$	SLIM	0.851	–	0.917	–	0.965	–
	DecisionTree	0.742	14.7%	0.832	10.2%	0.88	10.0%
	MEPFL(RF)	0.797	6.8%	0.856	7.1%	0.912	5.8%
	Ripper	0.723	17.7%	0.813	12.8%	0.856	12.7%

5.4 Ability to Deal with Imbalanced Datasets

5.4.1 Experimental Setup. To verify the performance advantages of our method in the data imbalance scenario, we extracted services to simulate the newly deployed services for the imbalance test. We selected the fault "network_delay" in docker005(dataset $\mathcal{A}$), the fault "OS Network" in apache02(dataset $\mathcal{B}$) and the fault "OS Network" in Tomcat01(dataset $\mathcal{B}$). Because these faults occurs more frequently than others in the entire dataset, making it easier to characterize the trend of the fault localization performance with the frequency of occurrence. We set the test set for each service to include two faults, and the training set gradually increases from one occurrence to *n*-2, where *n* is the total number of this faults.

5.4.2 Numerical Results. We show the results of each fault localization algorithms on three faults in Figure 9. Noticed that in this experiment, we removed some underperforming algorithms from the previous performance comparison. Because we couldn't determine whether the poor performance of diagnosing new services was due to the inherent shortcomings of the models or if it was a result of imbalanced data. From the Figure 9, we find out that our model is able to correctly identify all the testing faults when they occur in the training set for two times. While the rest algorithms need more training samples to produce reliable fault localization. It means that, given a newly deployed service, our proposed SLIM needs only a few historical data to train, thus can significantly reduce the numb fault the system needs to experience. We further balance the number of faults by upsampling the data of minority faults using SMOTE [14]. We report the results of each method with SMOTE in Figure 10. From Figure 10, we see that compared with the results in

Figure 9: The Imbalanced Dataset Experiment.
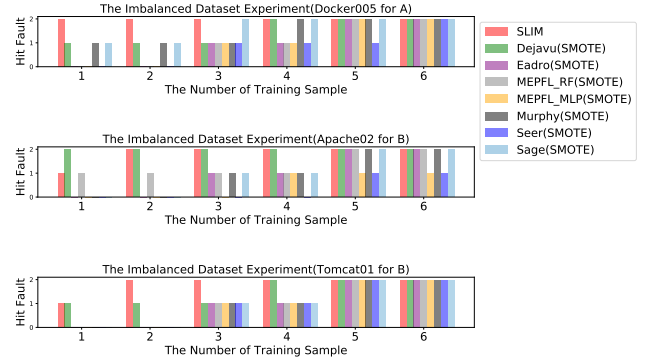
Figure 10: The Imbalanced Dataset Experiment with SMOTE.

Figure 9, few improvement is achieved when SMOTE is used for most algorithms. In Apache02 for Dataset $\mathcal{B}$, we find out that the Seer has a little promotion. However, the SMOTE produced negative impacts for Seer on fault "docker005" in Dataset $\mathcal{A}$. This is due to that SMOTE may introduce some noisy data to the model, which may affect the precision of Seer.

5.5 Performance of Knowledge Base Generator

We evaluate the generator using precision and time consumption. Two AIOPS experts are recruited to manually finish the knowledge base construction by their expert knowledge. The experts have longer than two years of operating experience and have publications at national conferences. As a baseline, we also construct the knowledge base using DecisionTree and Dejavu(local interpreter part). These are done using similar procedure to that used for SLIM. Table 6 shows the comparison results of our interpreter, expert knowledge and other baseline method for CloudRCA and MicroCBR using dataset A. Compared with the expert knowledge, our interpreter finishes the root cause analysis automatically and

loses just 5.1%–7.8% precision. Compared with other baseline methods, our interpreter's knowledge base improves precision by 7.8%-19.4%.

We also compare the time consumption by all methods in dataset A. Table 6 compares the results between the experts and the generator. In the experiment, experts require 5 hours to construct the knowledge base. Then, we compare the knowledge base effect among experts, our interpreter and other baseline interpreter. Compared with experts, our interpreter reduces the time required by 82% and has nearly the same precision.

Table 6: The Performance Comparison of Generator and Expert (Pre:Precision; TC:time-consuming)

Algorithm	Pre	Pre↑	TC	TC↓
CloudRCA(SLIM)	70%	–	57min	–
CloudRCA(DecisionTree)	60%	16.7%	46min	-24%
CloudRCA(Dejavu)	64%	7.8%	89min	36%
CloudRCA(Expert)	76%	-7.8%	5h	82%
MicroCBR(Interpreter)	74%	−%	55min	–
MicroCBR(DecisionTree)	62%	19.4%	43min	-28%
MicroCBR(Dejavu)	66%	11%	87	36.7%
MicroCBR(Expert)	78%	5.1%	5h	82%

6 LIMITATION AND FUTUREWORK

Due to the complexity of F1-Score in multi-class settings, our model, in order to trade off computational cost and performance, is optimized only for binary classification. This design choice makes our model not strictly end-to-end, which may decrease performance in the ultimate fault localization.

In future work, we aim to propose a new multi-class F1-Score optimization method to learn rule sets based on this module, achieving a fully end-to-end model. This approach seeks to address the performance trade-off issues encountered in the previous implementation.

7 CONCLUSION

In this paper, we propose an interpretable, effective and fast fault localization algorithm SLIM to directly optimize the F1 score, which is particularly applicable for highly imbalanced classification. Our experimental results demonstrate the superior performance and interpretability of SLIM in comparison with existing fault localization methods. In addition, the good adaptbility of SLIM makes it an ideal tool to handle large-scale microservice systems in many real-world scenarios involving frequent service change.

REFERENCES

[1] 2023. Dataset A and B. https://github.com/NetManAIOps/DejaVu. (2023).

[2] 2023. Dataset C. https://github.com/NetManAIOps/TraceRCA. (2023).

[3] 2023. Elasticsearch. https://www.elastic.co/. (2023).

[4] 2023. Grafana: The open observability platform. https://grafana.com/. (2023).

[5] 2023. High-quality, ubiquitous, and portable telemetry to enable effective observability. https://opentelemetry.io/. (2023).

[6] 2023. Jaeger: open source, end-to-end distributed tracing. https://www.jaegertracing.io/. (2023).

[7] 2023. Microsoft Doc. https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices. (2023).

[8] 2023. Prometheus. https://prometheus.io/. (2023).

[9] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. *arXiv preprint arXiv:2301.03797* (2023).

[10] Francis Bach et al. 2013. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends® in Machine Learning* 6, 2-3 (2013), 145–373.

[11] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems*. http://landing.google.com/sre/book.html

[12] Álvaro Brandón, Marc Solé, Alberto Huélamo, David Solans, María S Pérez, and Victor Muntés-Mulero. 2020. Graph-based root cause analysis for service-oriented and microservice architectures. *Journal of Systems and Software* 159 (2020), 110432.

[13] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002).

[14] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[15] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An empirical investigation of incident triage for online service systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 111–120.

[16] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous incident triage for large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 364–375.

[17] Lingchao Chen, Yicheng Ouyang, and Lingming Zhang. 2021. Fast and precise on-the-fly patch validation for all. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1123–1134.

[18] William W Cohen. 1995. Fast effective rule induction. In *Machine learning proceedings 1995*. Elsevier.

[19] Sanjeeb Dash, Oktay Gunluk, and Dennis Wei. 2018. Boolean Decision Rules via Column Generation. In *Advances in Neural Information Processing Systems*.

[20] Yang W Fengrui L. 2022. MicroCBR: Case-based Reasoning on Spatio-temporal Fault Knowledge Graph for Microservices Troubleshooting. In *International Conference on Case-Based Reasoning*.

[21] Yu Gan, Mingyu Liang, et al. 2021. Sage: Using Unsupervised Learning for Scalable Performance Debugging in Microservices. *arXiv preprint arXiv:2101.00267* (2021).

[22] Yu Gan, Yanqi Zhang, et al. 2019. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck (Eds.). ACM, 3–18. https://doi.org/10.1145/3297858.3304013

[23] Yu Gan, Yanqi Zhang, et al. 2019. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*. 19–33.

[24] Vipul Harsh, Wenxuan Zhou, Sachin Ashok, Radhika Niranjan Mysore, Brighten Godfrey, and Sujata Banerjee. 2023. Murphy: Performance Diagnosis of Distributed Cloud Applications. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 438–451.

[25] Chris Harshaw, Moran Feldman, Justin Ward, and Amin Karbasi. 2019. Submodular Maximization beyond Non-negativity: Guarantees, Fast Algorithms, and Applications. In *Proceedings of the 36th ICML*, Vol. 97. PMLR, 2634–2643.

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

[27] David R Hunter and Kenneth Lange. 2004. A tutorial on MM algorithms. *The American Statistician* 58 (2004).

[28] S Jegelka and J Bilmes. 2011. Submodularity beyond submodular energies: Coupling edges in graph cuts. In *Proceedings of the 2011 IEEE Conference on CVPR*.

[29] Yanjie Jiang, Hui Liu, Nan Niu, Lu Zhang, and Yamin Hu. 2021. Extracting concise bug-fixing patches from human-written patches in version control systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 686–698.

[30] Justin M. Johnson and Taghi M. Khoshgoftaar. 2019. Survey on Deep Learning with Class Imbalance. *Journal of Big Data* 6, 27 (2019).

[31] György Kovács. 2019. Smote-variants: A python implementation of 85 minority oversampling techniques. *Neurocomputing* 366 (2019), 352–354.

[32] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An End-to-End Troubleshooting Framework for Microservices on Multi-source Data. *arXiv preprint arXiv:2302.05092* (2023).

[33] Jon Lee, Vahab S Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. 2010. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM Journal on Discrete Mathematics* 23, 4 (2010).

[34] Xing Li, Yan Chen, Zhiqiang Lin, Xiao Wang, and Jim Hao Chen. 2021. Automatic policy generation for {Inter-Service} access control of microservices. In *30th USENIX Security Symposium (USENIX Security 21)*. 3971–3988.

[35] Zeyan Li, Junjie Chen, et al. 2021. Practical root cause localization for microservice systems via trace analysis. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 1–10.

[36] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, Yanhua Wang, Xu Du, Guoqing Duan, and Dan Pei. 2022. Actionable and Interpretable Fault Localization for Recurring Failures in Online Service Systems. In *Proceedings of the 2022 30th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2022)*.

[37] 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.

[38] Dewei Liu, Chuan He, Xin Peng, Fan Lin, Chenxi Zhang, Shengfang Gong, Ziang Li, Jiayu Ou, and Zheshun Wu. 2021. Microhecl: High-efficient root cause localization in large-scale microservice systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 338–347.

[39] Yiling Lou, Qihao Zhu, Jinhao Dong, Xia Li, Zeyu Sun, Dan Hao, Lu Zhang, and Lingming Zhang. 2021. Boosting coverage-based fault localization via graph-based representation learning. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 664–676.

[40] Meng Ma, Jingmin Xu, et al. 2020. Automap: Diagnose your microservice-based web applications automatically. In *Proceedings of The Web Conference 2020*. 246–258.

[41] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, et al. 2020. Diagnosing root causes of intermittent slow queries in cloud databases. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1176–1189.

[42] Tomasz Maciejewski and Jerzy Stefanowski. 2011. Local neighbourhood extension of SMOTE for mining imbalanced data. In *2011 IEEE symposium on CIDM*.

[43] Ajay Mahimkar. 2011. Rapid detection of maintenance induced changes in service performance. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*. 1–12.

[44] Leonardo Mariani, Mauro Pezzè, Oliviero Riganelli, and Rui Xin. 2020. Predicting failures in multi-tier distributed systems. *Journal of Systems and Software* 161 (2020), 110464.

[45] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica* 22, 3 (2012), 276–282.

[46] Sonu Mehta and Ranjita Bhagwan. 2020. Rex: Preventing bugs and misconfiguration in large services using correlated change analysis. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 435–448.

[47] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming* 14, 1 (1978), 265–294.

[48] Akshitha Sriraman and Thomas F Wenisch. 2018. μ suite: a benchmark suite for microservices. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 1–12.

[49] Li Wu, Johan Tordsson, Jasmin Bogatinovski, Erik Elmroth, and Odej Kao. 2021. Microdiag: Fine-grained performance diagnosis for microservice systems. In *2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence)*. IEEE, 31–36.

[50] Muhan Zeng, Yiqian Wu, Zhentao Ye, Yingfei Xiong, Xin Zhang, and Lu Zhang. 2022. Fault localization via efficient probabilistic modeling of program semantics. In *Proceedings of the 44th International Conference on Software Engineering*. 958–969.

[51] Yingying Zhang, Zhengxiong Guan, et al. 2021. CloudRCA: A root cause analysis framework for cloud computing platforms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 4373–4382.

[52] Nengwen Zhao, Junjie Chen, Zhaoyang Yu, et al. 2021. Identifying bad software changes via multimodal anomaly detection for online service systems. In *Proceedings of the 29th ACM Joint Meeting on the Foundations of Software Engineering(FSE)*. 527–539.

[53] Xiang Zhou, Xin Peng, et al. 2018. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Transactions on Software Engineering* 47, 2 (2018), 243–260.

[54] Xiang Zhou, Xin Peng, et al. 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 683–694.

A DETAIL PROCEDURE OF RULE GENERATION

We now present the details of our efficient rule generation method to solve the optimization problem (3) with objective (4) as Section 3.1.3 has briefly introduced. Let us rewrite the two logarithm components of (4) as $G(s) \triangleq \log(|\bigcup_{r \in s} \mathcal{X}_r^+|)$ and $C(s) \triangleq \log(|\bigcup_{r \in s} \mathcal{X}_r| + |\mathcal{X}^+|)$. As logarithm function is non-decreasing and concave, both $G(s)$ and $C(s)$ are non-negative monotone submodular functions [10]. Consequently, the objective function can be viewed as a difference between two submodular functions.

Objective reformulation. At each iteration, we try to find a rule r to maximize a distorted marginal gain of r with a parameter α. This α helps adjust the trade-off between precision and recall. The objective becomes

$$\max_{|r| \leq l} \quad \alpha G(r|s) - C(r|s), \tag{16}$$

where $G(r|s) \triangleq G(s \cup \{r\}) - G(s)$ and $C(r|s) \triangleq C(s \cup \{r\}) - C(s)$ denote the marginal gains of G and C when adding r to s. Because s is independent of r, (16) can be reduced to

$$\max_{|r| \leq l} \quad \alpha \log(|\mathcal{X}_r^+ \cup \mathcal{X}_s^+|) - \log(|\mathcal{X}_r \cup \mathcal{X}_s| + |\mathcal{X}^+|). \tag{17}$$

Notice that a rule r is a set of feature indices, so finding an optimal rule is equivalent to finding a set of features. The expression in (17) can be further rewritten as:

$$\begin{aligned} W(r) \triangleq &\alpha \log(|(\cap_{j \in r} \mathcal{X}_j)^+ \cup \mathcal{X}_s^+|) \\ &- \log(|(\cap_{j \in r} \mathcal{X}_j) \cup \mathcal{X}_s| + |\mathcal{X}^+|). \end{aligned} \tag{18}$$

Surrogate optimization of former term. Directly maximizing $W(r)$ is difficult. Although $|(\cap_{j \in r} \mathcal{X}_j)^+ \cup \mathcal{X}_s|$ is a supermodular function, the presence of logarithm function makes the property of $\log |(\cap_{j \in r} \mathcal{X}_j)^+ \cup \mathcal{X}_s|$ non-trivial. In our algorithm, $W(r)$ is maximized using MM algorithm [27], which iteratively increases the value of the objective function by maximizing a tight lower bound. We propose a proper lower bound of $W(r)$ by finding a lower bound of $\log |(\cap_{j \in r} \mathcal{X}_j)^+ \cup \mathcal{X}_s^+|$ and an upper bound of $\log(|(\cap_{j \in r} \mathcal{X}_j) \cup \mathcal{X}_s| + |\mathcal{X}^+|)$ separately.

To find a lower bound of the former term, $f(r) \triangleq |(\cap_{j \in r} \mathcal{X}_j)^+ \cup \mathcal{X}_s|$, it suffices to find a lower bound of $f(r)$, due to the non-decreasing property of the logarithm function. Nevertheless, the lower bound of $f(r)$ may be negative which makes the whole function undefined. Thus we introduce a surrogate function, ψ, whose domain is extended to $(-\infty, +\infty)$, while still maintaining the monotone property. We keep the function as its original value when $x \geq 1$, and when $x < 1$, we make the approximation by using the first-order Taylor expansion at the point $x = 1$, hence leading to that $\psi(x)$ equals to $\log(x)$ when $x \geq 1$, and equals to $x - 1$ when $x < 1$. $\log(f(r))$ and $\psi(f(r))$ share the same value for all r except

when $(\cap_{j \in r} \mathcal{X}_j)^+ = \emptyset$ and $s = \emptyset$, which corresponds to zero F1 score. Rule sets with zero F1 score can never be optimal as one can always construct another rule set with positive F1 score by selecting K positive samples and treating them as rules. As a result, the optimal solution that maximizes $\log(f(r))$ is equivalent to $\psi(f(r))$, which makes $\psi(f(r)))$ a tractable replacement to $\log(f(r))$.

Next we present the solution of finding a tight lower bound of $f(r)$. Notice that it is equivalent to finding a tight upper bound of the submodular function $-f(r)$, since $f(r)$ is supersubmodular. Motivated by the modular upper bounds of submodular functions presented in [28, 47], two proper lower bounds of $f(r)$ are given as

$$\begin{aligned} L^1_{f,r^{(t)}}(r) \triangleq &f(r^{(t)}) - \sum_{j \in r^{(t)} \setminus r} f(j|r^{(t)} \setminus \{j\}) + \sum_{j \in r \setminus r^{(t)}} f(j|\emptyset) \\ &\leq f(r), \ \forall \ r \subseteq \Gamma, \end{aligned} \tag{19}$$

$$\begin{aligned} L^2_{f,r^{(t)}}(r) \triangleq &f(r^{(t)}) - \sum_{j \in r^{(t)} \setminus r} f(j|\Gamma \setminus \{j\}) + \sum_{j \in r \setminus r^{(t)}} f(j|r^{(t)}) \\ &\leq f(r), \ \forall \ r \subseteq \Gamma, \end{aligned} \tag{20}$$

where $r^{(t)}$ denotes the current estimation of r. These two inequalities hold for all possible $r^{(t)}$, and the equality is achieved when $r = r^{(t)}$.

LEMMA 2. *Both $L^1_{f,r^{(t)}}(r)$ and $L^2_{f,r^{(t)}}(r)$ are modular and non-increasing functions.*

PROOF. As $L^1_{f,r^{(t)}}(r)$ and $L^2_{f,r^{(t)}}(r)$ share similar structure, then we only prove the results on $L^1_{f,r^{(t)}}(r)$, and that of $L^2_{f,r^{(t)}}(r)$ can be proved similarly. For $L^1_{f,r^{(t)}}(r)$, the marginal gain of adding element j into r is

$$\begin{aligned} L^1_{f,r^{(t)}}(j|r) &= L^1_{f,r^{(t)}}(r \cup \{j\}) - L^1_{f,r^{(t)}}(r) \\ &= \begin{cases} f(j|r^{(t)} \setminus \{j\}) & j \in r^{(t)} \\ f(j|\emptyset) & j \notin r^{(t)} \end{cases}. \end{aligned} \tag{21}$$

Note that $L^1_{f,r^{(t)}}(j|r)$ is independent with r and non-positive due to $f(r)$ being a non-increasing function. Then we can conclude that $L^1_{f,r^{(t)}}(r)$ is a modular and non-increasing function. □

Since we have obtained two lower bounds of $f(r)$, the lower bounds of $\psi(f(r))$ can be readily obtained as $\psi(L^1_{f,r^{(t)}}(r))$ and $\psi(L^2_{f,r^{(t)}}(r))$.

LEMMA 3. *Both $\psi(L^1_{f,r^{(t)}}(r))$ and $\psi(L^2_{f,r^{(t)}}(r))$ are submodular functions.*

PROOF. We only provide proof for function $\psi(L^1_{f,r^{(t)}}(r))$, as the result on $\psi(L^2_{f,r^{(t)}}(r))$ can be arrived at similarly. We

first prove that the function $\Psi(x) \triangleq \psi(x+\Delta) - \psi(x)$ is non-increasing for all non-negative Δ. The derivative of $\Psi(x)$ can be computed as

$$\Psi'(x) = \begin{cases} \frac{1}{x+\Delta} - \frac{1}{x} & x \geq 1 \\ \frac{1}{x+\Delta} - 1 & x+\Delta \geq 1, x < 1 \\ 0 & x+\Delta < 1 \end{cases}. \qquad (22)$$

It is clear that $\Psi'(x)$ is non-positive, which implies that $\Psi(x)$ is a non-increasing function. In other words, we have $\psi(x+\Delta) - \psi(x) \leq \psi(y+\Delta) - \psi(y)$ for all $x \geq y$. According to Lemma 2, we have $L^1_{f,r^{(t)}}(r \cup \{i\}) \geq L^1_{f,r^{(t)}}(r \cup \{i,j\})$ and $L^1_{f,r^{(t)}}(r)$ is a modular function. Then let $x = L^1_{f,r^{(t)}}(r \cup \{i\})$, $y = L^1_{f,r^{(t)}}(r \cup \{i,j\})$, and $\Delta = L^1_{f,r^{(t)}}(r) - L^1_{f,r^{(t)}}(r \cup \{i\}) = L^1_{f,r^{(t)}}(r \cup \{j\}) - L^1_{f,r^{(t)}}(r \cup \{i,j\}) \geq 0$, we have

$$\begin{aligned} \psi(x+\Delta) - \psi(x) &= \psi(L^1_{f,r^{(t)}}(r)) - \psi(L^1_{f,r^{(t)}}(r \cup \{i\}) \\ &\leq \psi(y+\Delta) - \psi(y) \\ &= \psi(L^1_{f,r^{(t)}}(r \cup \{j\})) - \psi(L^1_{f,r^{(t)}}(r \cup \{i,j\})), \end{aligned}$$

which shows that $\psi(L^1_{f,r^{(t)}}(r))$ is submodular. $\square$

Surrogate optimization of the latter term Next, we show how to find an upper bound of $\log(g(r))$, where $g(r) \triangleq |(\cap_{j \in r} \mathcal{X}_j) \cup \mathcal{X}_s| + |\mathcal{X}^+|$. Similar to $f(r)$, $g(r)$ is also a supermodular function and we proceed to find a tight lower bound of $-g(r)$, as [28, 47] also present a permutation-based method to find lower bounds for submodular functions. However, it is also mentioned in [28] that to certify an optimum of such a lower bound, one needs to examine about $O(m)$ permutations which is critical to the algorithm, where m is the size of the ground set, i.e. in our setting, $|\Gamma|$. Hence, in this paper, we employ another strategy to find an upper bound of $\log(g(r))$ by utilizing the concavity of logarithm functions. As $\log(x) \leq \log(x_0) + \frac{1}{x_0}(x - x_0)$, then a tight upper bound of $\log(g(r))$ is readily given as, which holds for any $r^{(t)}$,

$$\log(g(r)) \leq \log(g(r^{(t)})) + \frac{1}{g(r^{(t)})}(g(r) - g(r^{(t)})). \quad (23)$$

Since such an upper bound removes the non-linearity of the logarithm function, it shares the same property of $g(r)$, which is supermodular. We combine the bounds obtained above, and derive two tight lower bounds of $W(r)$ as follows,

$$W(r) \geq \alpha \psi(L^1_{f,r^{(t)}}(r)) - \frac{g(r)}{g(r^{(t)})} = V^1_{\alpha,r^{(t)}}(r), \quad (24)$$

$$W(r) \geq \alpha \psi(L^2_{f,r^{(t)}}(r)) - \frac{g(r)}{g(r^{(t)})} = V^2_{\alpha,r^{(t)}}(r). \quad (25)$$

Surrogate optimization of objective. The problem of maximizing $W(r)$ is translated to maximizing $V^1_\alpha(r|r^{(t)})$ and $V^2_\alpha(r|r^{(t)})$.

LEMMA 4. *Both* $V^1_\alpha(r|r^{(t)})$ *and* $V^2_\alpha(r|r^{(t)})$ *are non-monotone submodular functions.*

PROOF. Both $L^1_{f,r^{(t)}}(r)$ and $L^2_{f,r^{(t)}}(r)$ are submodular, and $g(r)$ is supermodular, so both $V^1_\alpha(r|r^{(t)})$ and $V^2_\alpha(r|r^{(t)})$ are submodular functions. It is obvious that $L^1_{f,r^{(t)}}(r), L^2_{f,r^{(t)}}(r)$ and $g(r)$ are all non-increasing functions, then $V^1_\alpha(r|r^{(t)})$ and $V^2_\alpha(r|r^{(t)})$ are both non-monotone functions. $\square$

B THE GUARANTEE THEOREM FOR OUR METHOD

THEOREM 5. *If* r^* *in line 4 of Algorithm 1 is the optimal solution of problem (17), Algorithm 1 will return a rule set s satisfying*

$$G(s) - C(s) \geq (1 - e^{1-\gamma})G(s^*) - C(s^*), \ |s| \leq K, \quad (26)$$

where s^* *represents the optimal solution of problem (4).*

We give the proof of the theorem as following:

PROOF. Define $q \triangleq \frac{1-\gamma}{K}$ and $\alpha_i \triangleq (1-q)^{K-(i+1)}$. Obviously, $\alpha_{-1} = (1-q)^K \leq e^{\gamma-1}$ and $\alpha_{K-1} = (1-q)^0 = 1$. We then define

$$\Phi_i(T) \triangleq \alpha_{i-1}G(T) - C(T), \quad (27)$$

$$\Psi_i(S_i, e) \triangleq \max(0, \alpha_i G(e|S_i) - C(e|S_i)). \quad (28)$$

Then we have

$$\begin{aligned} &\Phi_{i+1}(S_{i+1}) - \Phi_{i+1}(S_i) \\ &= \alpha_i G(S_{i+1}) - C(S_{i+1}) - \alpha_{i-1}G(S_i) + C(S_i) \\ &= \alpha_i G(S_{i+1}) - (1-q)\,\alpha_i G(S_i) - C(e|S_i) \\ &= \alpha_i(G(S_{i+1}) - G(S_i)) + q\alpha_i G(S_i) - C(e|S_i) \\ &\geq \Psi_i(S_i, e_i) + q\alpha_i G(S_i). \end{aligned} \quad (29)$$

Thus, we can give a lower bound of $\Psi_i(S_i, e_i)$ as follows:

$$\begin{aligned} &\Psi_i(S_i, e_i) \\ &\geq \alpha_i G(e_i|S_i) - C(e_i|S_i) \\ &\geq \frac{1}{K}\sum_{e \in OPT} \alpha_i G(e|S_i) - C(e|S_i) \\ &\geq \frac{1}{K}\alpha_i(G(OPT) - G(S_i)) - \frac{1}{K}\sum_{e \in OPT} C(e|S_i) \quad (30) \\ &\geq \frac{1}{K}\alpha_i(G(OPT) - G(S_i)) - \frac{1}{K(1-\gamma)}C(OPT), \quad (31) \end{aligned}$$

where (30) follows from the submodularity and monotone of $G(S)$, i.e.,

$$\sum_{e \in OPT} G(e|S_i) \geq G(OPT \cup S_i) - G(S_i) \geq G(OPT) - G(S_i).$$

And (31) follows from the definition of γ, i.e.,

$$\frac{C(e|OPT \setminus e)}{C(e|\emptyset)} \geq \frac{C(e|\Omega \setminus e)}{C(e|\emptyset)} \geq 1 - \gamma, \qquad (32)$$

and

$$\sum_{e \in OPT} C(e|S_i) \leq \sum_{e \in OPT} C(e|\emptyset)$$

$$\leq \sum_{e \in OPT} \frac{1}{(1-\gamma)} C(e|OPT \setminus e)$$

$$\leq \frac{1}{(1-\gamma)} C(OPT). \qquad (33)$$

Next we prove the bound of *DistortedGreedy* as follows. Note that the following equation holds:

$$\Phi_0(S_0) = \alpha_{-1} G(\emptyset) - C(\emptyset) = 0, \qquad (34)$$

$$\Phi_K(S_K) = \alpha_{k-1} G(S_K) - C(S_K) = G(S_K) - C(S_K). \qquad (35)$$

Then we have

$$G(S_K) - C(S_K) = \Phi_K(S_K) - \Phi_0(S_0)$$

$$= \sum_{i=0}^{K-1} \Phi_{i+1}(S_{i+1}) - \Phi_i(S_i)$$

$$\geq \sum_{i=0}^{K-1} \Psi_i(S_i, e_i) + q\alpha_i G(S_i)$$

$$\geq \sum_{i=0}^{K-1} (1-\gamma)\Psi_i(S_i, e_i) + q\alpha_i G(S_i) \qquad (36)$$

$$\geq \sum_{i=0}^{K-1} (1-\gamma)\left(\frac{1}{K}\alpha_i(G(OPT) - G(S_i))\right.$$

$$\left. - \frac{1}{K(1-\gamma)}C(OPT)\right) + q\alpha_i G(S_i)$$

$$\geq \sum_{i=0}^{K-1} q\alpha_i G(OPT) - \frac{1}{K}C(OPT)$$

$$\geq \left(1 - (1-q)^K\right) G(OPT) - C(OPT)$$

$$\geq (1 - e^{\gamma-1})G(OPT) - C(OPT) \qquad (37)$$

where (36) follows from $\Psi_i(S_i, e_i) \geq 0$ and $0 \leq (1 - \gamma) \leq 1$. This completes the proof. □

Theorem 5 shows that although a greedy updating procedure is employed, Algorithm 1 can still return a decent rule set. Notice that, since the curvature γ is usually unknown in advance, we run the proposed Algorithm 1 multiple times with different γ and return the rule set with best performance.