Mining Resource-Operation Knowledge to Support Resource Leak Detection

Chong Wang
Fudan University
Shanghai, China
wangchong20@fudan.edu.cn

Yiling Lou*
Fudan University
Shanghai, China
yilinglou@fudan.edu.cn

Xin Peng
Fudan University
Shanghai, China
pengxin@fudan.edu.cn

Jianan Liu
Fudan University
Shanghai, China
21210240250@m.fudan.edu.cn

Baihan Zou
Fudan University
Shanghai, China
bhzou21@m.fudan.edu.cn

ABSTRACT

Resource leaks, which are caused by acquired resources not being released, often result in performance degradation and system crashes. Resource leak detection relies on two essential components: identifying potential Resource Acquisition and Release (RAR) API pairs, and subsequently analyze code to uncover instances where the corresponding release API call is absent after an acquisition API call. Yet, existing techniques confine themselves to an *incomplete* pair pool, either pre-defined manually or mined from project-specific code corpus, thus limiting coverage across libraries/APIs and potentially overlooking latent resource leaks.

In this work, we propose to represent resource-operation knowledge as **abstract resource acquisition/release operation pairs** (*Abs-RAR pairs* for short), and present a novel approach called MIROK to mine such Abs-RAR pairs to construct a better RAR pair pool. Given a large code corpus, MIROK first mines Abs-RAR pairs with rule-based pair expansion and learning-based pair identification strategies, and then instantiates these Abs-RAR pairs into concrete RAR pairs. We implement MIROK and apply it to mine RAR pairs from a large code corpus of 1,454,224 Java methods and 20,000 Maven libraries. We then perform an extensive evaluation to investigate the mining effectiveness of MIROK and the practical usage of its mined RAR pairs for supporting resource leak detection. Our results show that MIROK mines 1,313 new Abs-RAR pairs and instantiates them into 6,314 RAR pairs with a high precision (i.e., 93.3%). In addition, by feeding our mined RAR pairs, existing approaches detect more resource leak defects in both online code examples and open-source projects.

CCS CONCEPTS

• **Software and its engineering** → *Software libraries and repositories*.

*Y. Lou is the corresponding author.

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12…$15.00
https://doi.org/10.1145/3611643.3616315

KEYWORDS

resource leaks, defect detection, knowledge representation, knowledge mining

ACM Reference Format:
Chong Wang, Yiling Lou, Xin Peng, Jianan Liu, and Baihan Zou. 2023. Mining Resource-Operation Knowledge to Support Resource Leak Detection. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3611643.3616315

1 INTRODUCTION

Resource leaks, which are caused by acquired resources not being released (*e.g.,* unclosed file handle), is a serious software defect that may cause runtime exceptions or program crashes. Resource leaks are prevalent in both software projects [13] and online code examples (e.g., even those code snippets accepted as correct answers in Stack Overflow posts [54]).

To date, researchers have proposed various automated resource leak detection techniques, which mainly rely on two essential components. First, identify the potential pairs of the Resource Acquisition API method and the corresponding Resource Release API method (*RAR pairs* for short); Then based on the RAR pairs, analyze the code to check whether the release API is not subsequently called after the acquisition API. For example, the resource acquisition API method *LockManager.acquireLock()* and the resource release API method *LockManager.releaseLock()* are one RAR pair of the lock resource; and a resource leak occurs when *LockManager.releaseLock()* is not called subsequently after *LockManager.acquireLock()*.

Although achieving promising effectiveness, the majority of existing resource leak detection techniques are concentrated on proposing more precise and more scalable code analysis approaches [21, 44, 49], while few of them focus on building a more *complete* RAR pair pool. However, knowing the RAR pair is the prerequisite of detecting the corresponding resource leak in the code, and an incomplete RAR pair pool would limit the effectiveness of the following code analysis. In particular, most existing techniques [44, 49] rely on human expertise and heuristic rules to predefine RAR pairs. Such predefined RAR pairs not only require non-trivial human efforts but also have limited coverage in libraries and APIs. For example, FindBugs [2] only considers the predefined *stream* related RAR pairs and thus could only detect *stream* related resource

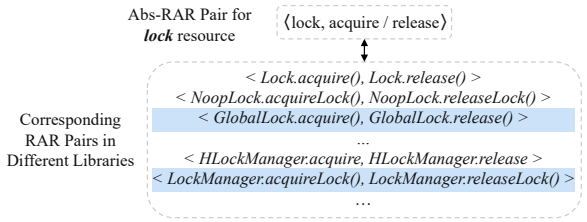

Figure 1: Abs-RAR Pair and RAR Pairs for Resource *lock*

leaks. More recently, Bian *et al.* [8] propose to identify RAR pairs for a given project by mining and classifying frequent API pairs within the project code corpus. However, such project-specific frequent mining is limited to specific libraries/projects and could only find RAR pairs that are frequently used in the project code corpus, missing those infrequent RAR pairs.

Thus, in this work, we aim at building a large RAR pair pool from a mass of projects and libraries, which could be used to enhance and support resource leak detection. In fact, it is challenging to build such a relatively complete RAR pair pool that precisely includes as many RAR pairs as possible. First, given the large variety of libraries and APIs in the wild, it is not easy to cover the diverse resources and acquisition/release operations. For example, just for the *lock* resource, there are over 738 relevant RAR pairs (reported by our approach), e.g., <*LockManager.acquireLock()*, *LockManager.releaseLock()*> and <*SoftLock.lock()*, *SoftLock.unlock()*>. In addition to popular resources, there are many less common resources (e.g., "semaphore"). Second, it is not easy to precisely identify whether two APIs belong to one RAR pair, since a pair of antisense verbs do not always indicate the acquisition and release operations. For example, although "open" and "close" are a pair of antisense verbs that are commonly used in acquisition and release APIs for some resources (*e.g.,* <*Database.open()*, *Database.close()*> and <*Connection.open()*, *Connection.close()*>), they do not denote resource acquisition and release in some context (*e.g.,* "openTag()" and "closeTag()" actually denote the start tag and end tag of an XML element).

To this end, we propose to represent resource-operation knowledge as **abstract resource acquisition/release operation pairs** (*i.e., Abs-RAR pairs* for short), and mine such Abs-RAR pairs from a large code corpus. Different from RAR pairs that are represented by concrete API methods, an Abs-RAR pair uses conceptual-level noun and verbs to describe the resource object and acquisition/release operations, which thus is able to represent a group of RAR pairs that share similar semantics. For example, as shown in Figure 1, the Abs-RAR pair ⟨lock, acquire / release⟩ could represent a set of RAR pairs that use "acquire" and "release" to manage the "lock" resource, such as <*GlobalLock.acquire()*, *GlobalLock.release()*> in library *xmlbeans* and <*LockManager.acquireLock()*, *LockManager.releaseLock()*> in library *copper-coreengine* (highlighted in Figure 1). Our insight is that abstract representation could extract more general resource-operation knowledge from a large code corpus and cover more diverse RAR pairs across different libraries/projects.

Based on this idea, we propose MIROK, a novel approach for **Mi**ning **R**esource **O**peration **K**nowledge, which constructs a large RAR pair pool to support resource leak detection. Given a large code corpus, MIROK first mines resource operation knowledge in the form of Abs-RAR pairs and then instantiates these Abs-RAR pairs into concrete RAR pairs. In particular, MIROK iteratively mines new Abs-RAR pairs based on existing ones with two strategies, i.e., the rule-based Abs-RAR pair expansion strategy derives new Abs-RAR pairs from existing ones based on the conceptual specialization relationships between resources, and the learning-based Abs-RAR pair identification strategy trains a sequence labeling model to identify new Abs-RAR pairs. After the Abs-RAR pairs are mined, MIROK then instantiates them into concrete RAR pairs in different libraries with matching-based rules.

We implement MIROK and apply it to mine resource operation knowledge from a large code corpus of 1,454,224 Java methods and 20,000 Maven libraries. We then perform an extensive evaluation to investigate the mining effectiveness of MIROK and the usage of its mined RAR pairs for supporting resource leak detection. First, we investigate its mining effectiveness by checking the quality of its mined Abs-RAR pairs and the instantiated RAR pairs. In total, MIROK mines 1,313 new Abs-RAR pairs based on 26 seed pairs and 89.2% (1,171) of them are manually checked as correct; and all Abs-RAR pairs are then instantiated into 6,314 RAR pairs from 2,261 libraries and 93.3% of them are manually checked as correct. Second, we feed our mined RAR pairs to existing resource leak detection analysis approaches and study how they could boost resource leak detection. Given the prevalence of resource leak issues in both online code examples and open-source projects, we evaluate how our generated RAR pairs help resource leak detection in both scenarios. First, for the online code examples, we find that with our newly-mined RAR pairs as inputs, even a simplistic static analysis approach successfully detects 4.5× more resource leaks (*i.e.,* 761 resource leaks in total) from 46,389 online code examples with a high precision (73.4%). Second, for the open-source projects, we enhance the widely-used resource leak static detection tool Findbugs by enriching its initial RAR pool with our new RAR pairs. Our results show that on 10 Github projects, the original Findbugs detects 4 resource leaks while the Findbugs extended with our RAR pairs detects 3 more previously-unknown resource leaks. Among them one has been confirmed by developers as of the submission time. In summary, the results show both the high quality and practical usage of our mined RAR pairs.

In summary, this paper makes the following contributions:

- **A novel representation** that represents resource operation knowledge with Abs-RAR pairs. Such Abs-RAR pairs could represent a group of semantically-similar RAR pairs across different libraries/projects, and thus convey more general resource-operation knowledge of a large code corpus.
- **A novel mining approach MIROK** that constructs a large RAR pair pool to support resource leak detection. MIROK first learns to mine Abs-RAR pairs from a large code corpus in an iterative learning process and then instantiates these Abs-RAR pairs into concrete RAR pairs.
- **A large-scale and high-quality RAR pair pool** that contains 6,314 RAR pairs mined from 1,454,224 Java methods over 2,261 libraries. To the best of our knowledge, this is the largest RAR pair pool for Java resource leak detection. We would publicly release our RAR pairs, which could be incorporated by existing or future resource leak detection work.

- **An extensive evaluation** that investigates both the mining effectiveness and the practical usage of mined RAR pairs for supporting resource leak detection. The results show that MIROK successfully mines and instantiates a large number of RAR pairs with a high precision, and our mined RAR pairs further help existing resource leak detection approaches to find more resource leaks in both online code snippets and open-source projects.

2 RELATED WORK

Since our work mines resource acquisition and release API pairs to support resource leak detection, in this section, we discuss the related work on resource leak detection (Section 2.1) and API usage pattern mining (Section 2.2).

2.1 Resource Leak Detection

Automated resource leak detection techniques [11, 16, 21, 22, 25–27, 29, 30, 32, 39, 40, 42–44, 49, 50] have been proposed to detect whether some resource is not being released after its acquisition. Typically there are two important components for resource leak detection. First, identify the potential RAR pairs (the pair of the resource acquisition API method and the corresponding resource release API method); (2) then based on the RAR pairs, analyze the code to check whether the release API is not subsequently called after the acquisition API.

The majority of existing resource leak detection techniques are concentrated on the analysis part by proposing more precise and more scalable code analysis approaches [11, 21, 27, 40, 44, 49]. For example, Torlak *et al.* [44] combine intra-procedural analysis and inter-procedural analysis to enable more scalable and more accurate detection for system resource leaks (e.g., I/O stream and database connections); Wu *et al.* [49] propose an inter-procedural and callback-aware static analysis approach to detect resource leak in Android apps; Kellogg *et al.* [21] incorporate ownership transfer analysis, resource alias analysis, and obligation fresh to enable more precise analysis. Our work is *orthometric* to this line of work, since we focus on building a more diverse and large RAR pair pool and our generated RAR pairs could further be incorporated into existing resource leak detection techniques.

In fact, the RAR pairs used in most existing techniques [44, 49] are often predefined by human expertise and heuristic rules, which not only require non-trivial human efforts but also have limited coverage in libraries and APIs. For example, Torlak *et al.* [44] manually collect RAR pairs that are related to *stream* and *database* resources in JDK. In addition, FindBugs [2] only considers the predefined *stream* related RAR pairs and thus could only detect *stream* related resource leaks. More recently, Bian *et al.* [8] propose SinkFinder, which mines RAR pairs for a given project by mining and classifying frequent API pairs within the project code corpus. However, its mined RAR pairs are limited to specific libraries/projects and only include RAR pairs that are frequently used in the project code corpus, thus missing those infrequent RAR pairs. Different from SinkFinder, our work represents and mines resource-operation knowledge via a novel abstract representation (e.g., Abs-RAR pairs), based on which we build a large RAR pair pool from a large amount of diverse libraries/projects.

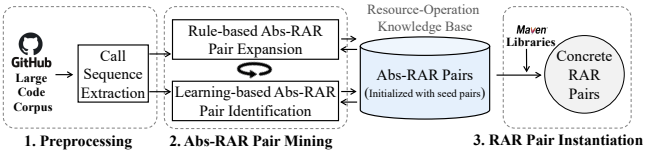

Figure 2: Approach Overview of MIROK

2.2 Mining API Usage Pattern for Misuse Detection

In other domains, there are also some techniques that mine API usage patterns to detect the relevant API misuse [7, 8, 10, 28, 36–38, 48, 53–55]. For example, Chang *et al.* [10] leverage frequent subgraph and itemset mining to mine rules for neglected condition detection. ExampleCheck [54] mines 180 API usage patterns for 100 popular Java APIs to detect API misuses such as missing control constructs and incorrect guard conditions. Our work targets a domain (i.e., resource leak) different from these techniques, i.e., we focus on mining the usage patterns of APIs on resource-operation knowledge. To this end, we not only propose a novel representation (i.e., Abs-RAR pairs) to represent the resource-knowledge-related API usage pairs, but also propose a novel learning-based mining approach to mine such Abs-RAR pairs from a large code corpus.

3 APPROACH

MIROK mines Abs-RAR pairs from a large code corpus and then instantiates the Abs-RAR pairs into concrete RAR pairs of different libraries. Figure 2 shows an overview of MIROK, which mainly consists of three phases: call sequence extraction, Abs-RAR pair mining, and RAR pair instantiation. (1) First, MIROK parses the large code corpus to extract method call sequences (Section 3.1). (2) Second, MIROK iteratively mines Abs-RAR pairs from the extracted sequences with two strategies (Section 3.2), i.e., rule-based Abs-RAR pair expansion and learning-based Abs-RAR pair identification. (3) Lastly, MIROK instantiates the mined Abs-RAR pairs into concrete RAR pairs (Section 3.3). In this work, we focus on resource-operation knowledge in Java.

3.1 Method Call Sequence Extraction

Since resource acquisition and releasing are often achieved by invoking relevant API methods, MIROK first parses the given code corpus to extract method call sequences as the input of the Abs-RAR pair mining.

For each source file in the code corpus, MIROK first parses it into an abstract syntax tree (AST) with the javalang toolkit [3], and then extracts a sequence of method calls based on AST node types following previous works [14, 15].

An extracted method call sequence S is a list of method calls ordered by their appearance in the code. For each method call $o.m$, o and m denote the name of the object and the called method respectively. For example, for the method *verify()* shown in Figure 3(a), MIROK extracts a method call sequence shown in Figure 3(b).

3.2 Abs-RAR Pair Mining

Abs-RAR Pair Definition. An Abs-RAR pair is a two-tuple P_{abs}= $\langle res, acq\,/\,rel\rangle$, where res refers to a resource concept, and acq and

```
void verify(File recoveryFile) {
    try {
        wakelock.acquire();
        RecoverySystem.verifyPackage(recoveryFile);
    } finally {
        wakelock.release();
    }
}
```

(a) Code Example

Call Sequence: *wakeLock.acquire* *RecoverySystem.verifyPackage* *wakeLock.release*

(b) Method Call Sequence

Token Sequence:	*wake*	*lock*	*acquire*	*recovery*	*system*	*verify*	*package*	*wake*	*lock*	*release*
Part Sequence:	*OBJ*	*OBJ*	*MD*	*OBJ*	*OBJ*	*MD*	*MD*	*OBJ*	*MD*	*MD*
Tag Sequence:	T_{res}	T_{res}	T_{acq}	T_{non}	T_{non}	T_{non}	T_{non}	T_{res}	T_{res}	T_{rel}

(c) Token Sequence, Part Sequence, and Tag Sequence

Figure 3: A Code Example and its Corresponding Call Sequence, Token Sequence, Part Sequence, and Tag Sequence

rel is a pair of *conceptual-level* resource acquisition and release operations for *res*. The benefits of conceptual-level abstraction in Abs-RAR are two-folds. First, an Abs-RAR pair can be regarded as an abstraction of similar RAR pairs that are implemented in different libraries. For example, ⟨lock, acquire / release⟩ is an Abs-RAR pair for *lock* resource, as shown in Figure 1, it could represent a group of relevant RAR pairs defined in different libraries. Second, representing the resources and acquisition/release operations in a conceptual way could further support resource generalization/specialization (*e.g.,* "wake lock" is a specialization of "lock"), synonyms (*e.g.,* "database" and "db" are synonyms), and semantic relevance (*e.g.,* "lock" and "semaphore" are both concurrency-related resources, "acquire"/"release" and "lock"/"unlock" are both concurrency-related acquisition/release operations). These properties can help the transfer and generalization of resource-operation knowledge, thus are beneficial for the mining.

With a small set of seed Abs-RAR pairs, MIROK iteratively mines Abs-RAR pairs from the extracted method call sequences via two strategies (i.e., rule-based Abs-RAR pair expansion and learning-based Abs-RAR pair identification). Rule-based Abs-RAR pair expansion derives new Abs-RAR pairs from existing ones based on the conceptual specialization relationships between resources; and learning-based Abs-RAR pair identification trains a sequence labeling model based on existing Abs-RAR pairs and uses the model to identify new Abs-RAR pairs from the method call sequences. In each iteration, MIROK leverages both strategies to extract new Abs-RAR pairs, which are then included to extend the pool of Abs-RAR pairs. The iteration process repeats until the maximum number of iterations is reached or no new Abs-RAR pairs are found. We then detail each mining strategy respectively.

3.2.1 Rule-based Abs-RAR Pair Expansion. Rule-based Abs-RAR pair expansion derives new Abs-RAR pairs from existing ones based on the specialization relationships between resources. In particular, a conceptual specialization relationship means that a resource concept is a special instantiation of another resource concept. For example, we regard the resource concept "wake lock" as a specialization of the resource concept "lock", and it is very likely that the specialized resource concept "wake lock" shares the similar acquisition/release operations as "lock" (e.g., "acquire" and "release").

In particular, given a method call sequence *S*, MIROK first identify all the candidate method call pairs, which match any existing Abs-RAR pair as a conceptual specialization; then MIROK derives new Abs-RAR pairs based on the candidate pairs. We then introduce the detailed steps as follows.

First, MIROK tokenizes each method call $o.m$ in the method call sequence *S* based on camel case [14, 15, 31] and parse part-of-speech (POS) tags [17, 46, 47]. In this way, each method call $o.m$ is tokenized into the form [*O VB NP REST*], where *O* denotes the object name, and *VB*, *NP*, and *REST* denote the verb, noun phrase, and the rest tokens in *m*, respectively. For example, the method call *FilesController.openFileByName* is transformed into [*O("file controllers") VB("open") NP("file") REST("by name")*].

Second, for each two methods calls (i.e., $o.m_1$ and $o.m_2$) in *S*, MIROK identifies whether they are a pair of resource acquisition/release operations (based on our pairing rules) and whether they match any existing Abs-RAR pair ⟨*res*, *acq* / *rel*⟩ as a conceptual specialization (based on our matching rules). The qualified method call pairs are considered as candidate pairs.

- **Matching Rules.** We consider a method call $o.m$ with the form [*O VB NP REST*] matches *res.acq* (or *res.rel*) if it satisfies the following two conditions: (1) the noun phrase *NP* contains *res*, or *O* contains *res* and *NP* is empty; (2) the verb *VB* equals *acq* (or *rel*).
- **Pairing Rules.** We consider two method calls $o.m1$ and $o.m2$ as a pair of acquisition/release operations , if they satisfy the following three conditions: (1) $o.m_1$ and $o.m_2$ match *res.acq* and *res.rel* respectively; (2) the noun phrase (*i.e., NP*) and the rest (*i.e., REST*) in $o.m_1$ and $o.m_2$ are the same; and (3) $o.m_1$ appears before $o.m_2$ in the same method call sequence.

Third, based on each candidate pair $<o.m_1, o.m_2>$, MIROK creates a new Abs-RAR pair ⟨*res′*, *acq* / *rel*⟩, where *res′* is *o* if *o* contains *res* and *NP* otherwise. For example, given an Abs-RAR pair ⟨lock, acquire / release⟩, we can derive a candidate Abs-RAR pair ⟨wake lock, acquire / release⟩ from the method call sequence shown in Figure 3(b). To avoid generating an overwhelming number of Abs-RAR pairs in each iteration, we only include those ones whose frequency is more than three into the Abs-RAR pair pool.

3.2.2 Learning-based Abs-RAR Pair Identification. Rule-based pair expansion only mines new pairs which have conceptual specialization relationships of existing pairs. Hence, to include more diverse Abs-RAR pairs, MIROK further leverages a learning-based Abs-RAR pair identification strategy. In particular, MIROK treats the Abs-RAR pair identification problem as a sequence labeling problem [20], leverages existing Abs-RAR pairs to automatically label the data for model training, and then utilizes the trained model to extract new Abs-RAR pairs. We then discuss details on the problem definition, automatic data labeling, model design, and training/prediction procedure, respectively.

Problem Definition. We model Abs-RAR pair identification as a sequence labeling problem. Sequence labeling has been widely studied in natural language processing (NLP) [20] and applied to software engineering tasks, *e.g.,* entity/concept recognition [45, 51, 52], and its main goal is to tag each token in a token sequence.

In our context, we define four different tags, *i.e.,* T_{res}, T_{acq}, T_{rel}, and T_{non} (denoting tokens corresponding to resource, acquisition operation, release operation, and others, respectively). With each method call sequence as a token sequence, our goal is to assign a tag to each token; and based on the combination of these tagged tokens, we could further identify possible combinations of a resource *res*, an acquisition operation *acq*, and a release operation *rel* that can form an Abs-RAR pair like $\langle res, acq\,/\,rel\rangle$. For example, the token sequence and the tag sequence of the method call sequence in Figure 3(b) are shown in Figure 3(c).

Automatic Data Labeling. To prepare the training data for the sequence labeling model, MIROK automatically labels the method call sequences via a distant supervision method. Distant supervision was originally proposed for the data labeling problem in relation extraction [35]. The main idea is to use the existing knowledge base to automatically label training data. In our work, we use existing Abs-RAR pairs as the resource-operation knowledge base to automatically label method call sequences. For each method call sequence S, if it matches an existing Abs-RAR pair, it is automatically labeled and used a training sample; otherwise, it is used as a prediction sample for identifying new Abs-RAR pairs. We then explain the details as follows.

- *Token/Part Sequence Generation.* MIROK generates a token sequence and a part sequence for the method call sequence S. These two sequences will be used as the input of the sequence labeling model. For each method call $o.m$ in S, MIROK tokenizes o and m by camel case to get a subsequence of tokens. For example, the method call *"wakeLock.acquire"* is tokenized into a subsequence ["wake", "lock", "acquire"]. Then MIROK generates a token sequence for S by concatenating the token subsequences of all the method calls in S. To record the part of method call (object or method) to which a token belongs, MIROK generates a corresponding part sequence for the token sequence. For each token, the part sequence uses *OBJ* or *MD* to indicate that the token is from the object part or the method part of the method call. For example, for the above token subsequence ["wake", "lock", "acquire"], its corresponding part subsequence is [*OBJ*, *OBJ*, *MD*].
- *Matching Sequences with Abs-RAR Pairs.* MIROK then checks whether S could match any existing Abs-RAR pair. For each Abs-RAR pair, MIROK checks whether any pair of method calls in S matches the Abs-RAR pair based on the *pairing rules* in Section 3.2.1. If such a pair of method calls is found, the method call sequence S matches the Abs-RAR pair. If S matches two Abs-RAR pairs and one of them is derived from the other, we only keep the derived one, *i.e.,* the specialized one. For example, the method call sequence shown in Figure 3(b) matches both ⟨lock, acquire / release⟩ and ⟨wake lock, acquire / release⟩ and we only keep the latter.
- *Tag Sequence Generation.* If S matches an existing Abs-RAR pair $\langle res, acq\,/\,rel\rangle$, MIROK then generates a tag sequence for it. Assuming the corresponding method call pair in S is $<o.m_1, o.m_2>$, MIROK transforms $o.m_1$ and $o.m_2$ into the form [*O VB NP REST*] and generates a tag for each token in the token sequence of S in the following way: the verbs (*i.e., VB*) in m_1 and m_2 are tagged with T_{acq} and T_{rel} respectively; the tokens in o or the noun phrase (i.e., *NP*) in m_1 and m_2 that contain *res* are tagged with T_{res}; all the other tokens are tagged with T_{non}. For example, for the method call sequence shown in Figure 3(b) MIROK generates a tag sequence as shown in Figure 3(c).

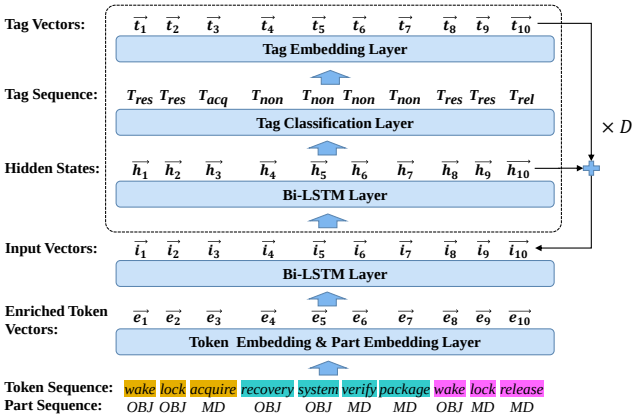

Figure 4: Sequence Labeling Model in MIROK

In this way, if a tag sequence could be generated for S, MIROK treats it as a training sample and uses its token sequence, part sequence, and tag sequence for training; otherwise, MIROK treats it as a prediction sample and conducts sequence labeling on its token sequence and part sequence to identify new Abs-RAR pairs.

Model Architecture. The sequence labeling model in MIROK needs to meet the following two requirements. First, the model should be able to produce multiple tag sequences for a method call sequence, as there might be multiple Abs-RAR pairs. Thus, we leverage the iterative grid labeling in OpenIE6 [24], which could support iteratively producing multiple tag sequences for a method call sequence. Second, the model should be able to incorporate both the textual semantics of tokens and the sequential information of method calls in the learning. Thus, we integrate LSTM (Long Short-Term Memory) [18] in the model, which is commonly used for capturing token semantics and sequential contexts.

Figure 4 shows the detailed architecture of our sequence labeling model, which takes a token sequence $[token_1, token_2, ..., token_N]$ and a part sequence $[part_1, part_2, ..., part_N]$ as input and outputs a fixed number (D) of tag sequences. Here, D indicates the depth of the model. Note that some or even all of the D produced tag sequences may include only the tag T_{non}, indicating that no Abs-RAR pairs are involved. Each token $token_i$ and its corresponding part $part_i$ are first projected into their vector representations $emb_{tok}(token_i)$ and $emb_{part}(part_i)$ respectively through an embedding layer. Then the two vectors are concatenated into an enriched token vector $\vec{e_i} = emb_{tok}(token_i) \oplus emb_{part}(part_i)$. After processing all the tokens, their enriched vectors, *i.e.,* $[\vec{e_1}, \vec{e_2}, ..., \vec{e_N}]$, are fed into a Bi-LSTM layer to obtain the corresponding input vectors $[\vec{i_1}, \vec{i_2}, ..., \vec{i_N}]$ for the iterative grid labeling block. The iterative grid labeling block includes the following three layers:

- *Bi-LSTM Layer.* This layer takes the input vectors and outputs hidden states $[\vec{h_1}, \vec{h_2}, ..., \vec{h_N}]$.
- *Fully-Connected Tag Classification Layer.* This layer predicts a probability distribution p_i of the four tags (*i.e.,* T_{res}, T_{acq}, T_{rel},

T_{non}) for each hidden state $\vec{h_i}$. Based on p_i, a tag sequence $[tag_1, tag_2, ..., tag_N]$ can be generated by selecting a tag tag_i that has the highest probability in p_i.

- *Tag Embedding Layer.* This layer converts the tag sequence into the corresponding vector sequence $[\vec{t_1}, \vec{t_2}, ..., \vec{t_N}]$, where $\vec{t_i}$ is the vector representation of tag_i.

After each block iteration, current hidden states $[\vec{h_1}, \vec{h_2}, ..., \vec{h_N}]$ and tag vectors $[\vec{t_1}, \vec{t_2}, ..., \vec{t_N}]$ are added to produce new input vectors, *i.e.,* $[\vec{h_1} + \vec{t_1}, \vec{h_2} + \vec{t_2}, ..., \vec{h_N} + \vec{t_N}]$, for the next iteration of the block. The iterative process continues until D tag sequences are generated for the current token sequence.

Token Embeddings Pre-training. To alleviate the parameter overfitting problem in training and the out-of-vocabulary problem in prediction, MIROK leverages token embeddings pre-trained on the entire code corpus to initialize the token embedding layer of the model. The token embeddings are pre-trained on all the token sequences extracted from the code corpus using Word2Vec [34]. The token embeddings can help capture the semantic associations between tokens by making tokens that frequently appear in similar contexts as close as possible in the space. For example, "lock", "semaphore", "acquire", and "release" are close in the space and the embeddings can help the sequence labeling model to capture the associations among the resources (*e.g.,* "lock", "semaphore") and acquisition/release operations (*e.g.,* "acquire", "release"). After initialized by the pre-trained token embeddings, the parameters of the token embedding layer are frozen during model training.

Model Training. According to the grid labeling architecture of the model, each training sample, which represents a method call sequence, needs to have exactly D (the depth of the model) tag sequences. Therefore, MIROK randomly selects D tag sequences for the training sample if it has no less than D tag sequences; otherwise, MIROK generates some tag sequences that include only the tag T_{non} for the training sample to reach the number D.

During training, the model parameters are continuously optimized with the objective of minimizing the loss. MIROK defines the loss function as the cross-entropy loss between the predicted tag sequences and the labeled ones. For each training sample, the loss is the sum of the loss in the D iterations of the grid labeling block. In the i-th iteration, the model predicts a tag sequence *pred* with a probability sequence *prob*. Here, the j-th element in *prob* is the predicted probability of the j-th tag in *pred*. Then MIROK takes the corresponding tag sequence *gold* (*i.e.,* the i-th tag sequence of the training sample) and calculates the cross-entropy loss $\mathcal{L}^{CE}$ between *pred* and *gold* based on *prob*.

However, the cross-entropy loss is not so sensitive to the difference between the acquisition operation (*i.e., acq*) and release operation (*i.e., rel*), as they are close in the token embedding space and have the same token part labels (*i.e., MD*). Therefore, we add a penalty term to the cross-entropy loss to better reflect the difference between the acquisition operation and the release operation of the same Abs-RAR pair. Given a predicted tag sequence *pred* and the corresponding tag sequence in the training sample *gold*, MIROK obtains the hidden states of the two tokens in the training sample corresponding to the *acq* and *rel* tags in *gold*. Based on the two hidden states (*i.e.,* $\vec{h_{acq}}$ and $\vec{h_{rel}}$) we use Equation 1 to calculate their cosine similarity and use it as the penalty term.

$$\mathcal{P} = \cos(\vec{h_{acq}}, \vec{h_{rel}}) \tag{1}$$

The final loss $\mathcal{L}$ is calculated as Equation 2, where $\mathcal{L}_i^{CE}$ and $\mathcal{P}_i$ are the cross-entropy loss and penalty term in the i-th block iteration. Based on the loss function, MIROK trains the model using a stochastic optimizer such as Adam [23].

$$\mathcal{L} = \sum_{i=1}^{D} (\mathcal{L}_i^{CE} + \mathcal{P}_i) \tag{2}$$

Before training, MIROK splits the training samples into a training set and a validation set by 9:1. The training set is used for model training and the validation set is used to validate the performance of the model. When the loss on the validation set (*i.e.,* validation loss) does not decrease in three consecutive epochs, MIROK stops the training process and chooses the version of the model having the lowest loss as the trained model.

Model Prediction and Abs-RAR Pair Extraction. MIROK use the trained model to predict tag sequences for each prediction sample and extracts new Abs-RAR pairs from the predicted tag sequences. Given a prediction sample S, MIROK takes its token sequence and part sequence as input and uses the trained model to predict D tag sequences for it. For each predicted tag sequence *pred*, MIROK tags the tokens in S according to the corresponding tags in *pred*. Then MIROK tries to extract an Abs-RAR pair $\langle res, acq\,/\,rel\rangle$ where *res*, *acq*, and *rel* are continuous tokens in S that respectively have the corresponding tags (*i.e.,* T_{res}, T_{acq}, T_{rel}). If such an Abs-RAR pair is extracted and *acq* and *rel* are not the same, MIROK treats it as a candidate Abs-RAR pair and calculates its confidence by averaging the probabilities of all the tags in *pred*. After all the prediction samples are processed, MIROK merges identical candidate Abs-RAR pairs and averages their confidences as the final confidence. To ensure the quality of new Abs-RAR pairs, we only include the candidate Abs-RAR pairs whose frequency is more than three and confidence is higher than 0.8 to extend the Abs-RAR pair pool.

3.3 RAR Pair Instantiation

The Abs-RAR pairs convey general resource-operation knowledge summarized from a large code corpus, and they could further be instantiated into concrete RAR pairs (*i.e.,* the pair of API methods) that are defined in different libraries. These concrete RAR pairs could be further incorporated by existing resource leak detection techniques. Different from existing work (*e.g.,* SinkFinder) mining RAR pairs from the project code that *uses* the APIs in RAR pairs, our RAR pairs are instantiated from the library code that directly *defines* the APIs in RAR pairs. Therefore, our approach could identify a comprehensive pool of RAR pairs, including those pairs that are infrequently or unpairwisely used in the project code.

In particular, for all API methods defined in a given library, MIROK regards any two API methods $c.m_1$ and $c.m_2$ (defined in a same class c) as an instantiation of the Abs-RAR pair $\langle res, acq\,/\,rel\rangle$, if $c.m_1$ and $c.m_2$ match $res.acq$ and $res.rel$ respectively. The matching rules for method calls and resource operations are the same as those in Section 3.2.1. For example, for the two API methods

acquire() and *release()* which are both defined in same class *GlobalLock* of the library *xmlbeans*, we consider them as an instantiated RAR pair of the Abs-RAR pair ⟨lock, acquire / release⟩.

4 IMPLEMENTATION

The implementation of MIROK includes the following important preparations and decisions.

Seed Abs-RAR Pair Collection. The seed Abs-RAR pairs used in our implementation are extracted from the API method pairs for resource acquisition and release used in Wu *et al.*'s work [49]. We manually extract 26 seed Abs-RAR pairs from the signatures of the API method pairs. For example, from the API method pair <*WifiManager.Lock.acquire()*, *WifiManager.Lock.release()*> we extract an Abs-RAR pair ⟨lock, acquire / release⟩. The full list of the 26 seed Abs-RAR pairs can be found in our replication package [6].

Dataset Construction. (1) The code corpus used for Abs-RAR pair mining mainly consists of two sources. First, we obtain crawl the open-source Java projects on GitHub that were created during the period of 2010 to 2016 and have more than 50 stars. These projects have about 4M methods. Second, we collect the data in CodeSearchNet [19], which is a collection of datasets and benchmarks for code retrieval. It contains about 500K Java methods. We merge the two data sources and filter out duplicate methods. We further filter out the methods that contain less than three method calls, as they are unlikely to include RAR pairs. We also filter out the methods that contain more than 50 method calls, since the training of the sequence labeling model will consume a lot of computational resources when processing too long call sequences. As a result, the final code corpus contains 1,454,224 Java methods. (2) The libraries used for RAR pair instantiation include top 20,000 Maven libraries in the Libraries.io dataset [4] according to their stars on GitHub.

Maximum Number of Iterations in Abs-RAR Pair Mining. We set the maximum number of iterations to 20 based on our observation that the iterative mining process usually reaches convergence in around 20 iterations (see Section 5.5).

Model Construction. The sequence labeling model is implemented using PyTorch 1.9.1 [5], which is one of the most popular machine learning frameworks. The hyper-parameters of the model are as follows. The sizes of token embeddings and part embeddings in the embedding layer are 100 and 30 respectively. The hidden sizes of the two Bi-LSTM layers are both 64. The tag classification layer consists of two layers of fully-collected networks whose hidden size is 128. The settings of these hyper-parameters are based on common practice in related work and the considerations of machine configuration and training overhead. The depth of the model (*i.e., D*) is set to 2.

Model Training. We add a dropout of 0.5, a commonly used technique for regularization, between the second Bi-LSTM layer and the tag classification layer. The learning rate is set to 0.001 and the training batch size is set to 64. The maximum number of training epochs is set to 100, which is rarely reached due to early stopping.

5 EVALUATION

We apply MIROK to first mine Abs-RAR pairs from a large code corpus of 1,454,224 Java methods and then to instantiate the Abs-RAR pairs into concrete RAR pairs for 20,000 Maven libraries. In this section, we first evaluate the mining effectiveness of MIROK by investigating the quality of its mined Abs-RAR pairs (RQ1) and its instantiated RAR pairs (RQ2); we then evaluate the practical usage of its RAR pairs by investigating whether they could boost resource leak detection in online code examples (RQ3.a) and open-source projects (RQ3.b); we also perform an ablation study to investigate the contribution of both mining strategies (RQ4).

Table 1: Some Examples of the Mined Abs-RAR Pairs

Domain	Abs-RAR Pairs
Concurrency	⟨mutex, lock / unlock⟩, ⟨semaphore, acquire / release⟩
Database	⟨database, connect / disconnect⟩, ⟨db, connect / disconnect⟩
File	⟨xml, open / close⟩, ⟨zip, <init> / close⟩
I/O	⟨stream, <init> / close⟩, ⟨reader, <init> / close⟩
Web/Network	⟨socket, connect / close⟩, ⟨client, create / destroy⟩
Device	⟨camera, start / stop⟩, ⟨device, open / close⟩
Service	⟨manager, activate / deactivate⟩, ⟨compactor, initialize / close⟩

- **RQ1. (Effectiveness of Abs-RAR Pair Mining)**: How many Abs-RAR pairs are mined by MIROK? How many of them are valid?
- **RQ2. (Effectiveness of RAR Pair Instantiation)**: How many concrete RAR pairs are instantiated from the mined Abs-RAR pairs? How many of them are valid?
- **RQ3. (Practical Usage for Resource Leak Detection):**
  - **RQ3.a (Resource Leaks in Online Code Examples)**: Can our mined RAR pairs help detect resource leaks in online code examples?
  - **RQ3.b (Resource Leaks in Open-source Projects)**: Can our mined RAR pairs help detect resource leaks in open-source project?
- **RQ4. (Impact of Each Mining Strategy)**: What is the contribution of the each mining strategy in MIROK?

5.1 RQ1: Abs-RAR Pair Mining

In this RQ, we analyze the Abs-RAR pairs mined by MIROK.

5.1.1 Protocol. We manually assess the quality of the Abs-RAR pairs mined by MIROK. In particular, we invite two developers who have more than four years Java development experience to independently inspect the validity of all the mined Abs-RAR pairs. For each Abs-RAR pair, they are asked to annotate whether it is valid, i.e., representing a valid pair of acquisition/release operations on a resource. To make a proper decision, they can search and consult external sources for help, e.g., reading various technical documents on Google or search GitHub projects using specific keywords to find code snippets that include the Abs-RAR pair. If their annotations for an Abs-RAR pair are inconsistent, a third annotator is assigned to give an additional annotation to resolve the conflict with a majority-win strategy. The Cohen's Kappa agreement [33] of the two annotators is 0.812 in our manual assessing, indicating a substantial agreement.

5.1.2 Results. Based on the 26 seed Abs-RAR pairs, MIROK mines 1,313 new Abs-RAR pairs from the code corpus, among which 1,171 (89.2%) are confirmed to be valid. These new Abs-RAR pairs involve 982 resources (964 of them are not included in the seed pairs) and 43 operation pairs (30 of them are not included in the seed pairs).

Table 1 shows some examples of the valid Abs-RAR pairs. The involved resources scatter in different domains such as concurrency, database, file, I/O, web/network, device, and service. These Abs-RAR pairs reflect the following capabilities of MIROK in identifying new Abs-RAR pairs based on existing ones.

1) Conceptual Specialization of Resources. New Abs-RAR pairs perform the same acquisition/release operations on conceptually specialized resources. For example, ⟨wake lock, acquire / release⟩ has the same acquisition/release operations with ⟨lock, acquire / release⟩ and "wake lock" is the conceptual specialization of "lock". This strategy is directly used by the rule-based Abs-RAR pair expansion (see Section 3.2.1).

2) Conceptually Relevant Resources. New Abs-RAR pairs perform the same acquisition/release operations on conceptually relevant resources. For example, ⟨semaphore, acquire / release⟩ has the same acquisition/release operations with ⟨lock, acquire / release⟩ and "semaphore" is a conceptually relevant concept of "lock".

3) New Acquisition and Release Operations on Existing Resources. New Abs-RAR pairs perform different acquisition/release operations on existing resources. For example, ⟨manager, activate / deactivate⟩ has the same resource with ⟨manager, open / close⟩ and involves new acquisition/release operations "activate"/"deactivate" instead of "open"/"close".

4) New Combinations of Existing Resources and Operations. New Abs-RAR pairs combine existing resources and acquisition/release operations in different ways. For example, ⟨database, connect / disconnect⟩ combines the resource "database" and the acquisition/release operations "connect"/"disconnect" which exist in existing Abs-RAR pairs such as ⟨database, open / close⟩ and ⟨connection, connect / disconnect.⟩

5) Completely New Resources and Operations. New Abs-RAR pairs involve completely new resources and acquisition/release operations. For example, ⟨compactor, initialize / close⟩ and ⟨client, create / destroy⟩ both involve unseen resources and acquisition/release operations.

In summary, MIROK can mine a wide variety of Abs-RAR pairs that involve new resources and/or acquisition/release operations. The results reflect the capabilities of MIROK in learning the latent relationships between resources and acquisition/release operations.

We also investigate the invalid Abs-RAR pairs mined by MIROK, and find that most of them are caused by the invalid combinations of resources and acquisition/release operations. The problem might be caused by the accumulation of errors during the iterative learning process, since MIROK adopts a distant supervision method to train the sequence labeling model. For example, if an invalid Abs-RAR pair (e.g., ⟨stream, mark / reset⟩) is mined, it might lead to more invalid Abs-RAR pairs with other I/O related resources (*e.g.*, "buffer") in subsequent iterations (e.g., ⟨buffer, mark / reset⟩). To address this issue, future work could further introduce perturbation to mitigate the negative affects in each iteration.

Summary: MIROK is highly effective in mining Abs-RAR pairs by mining 1,313 new Abs-RAR pairs with 89.2% valid rate. The mined Abs-RAR pairs cover a wide spectrum of unseen resources and/or acquisition/release operations from different domains.

5.2 RQ2: RAR Pair Instantiation

In this RQ, we analyze the concrete RAR pairs instantiated by MIROK for 20,000 Maven libraries.

5.2.1 Protocol. We introduce the RAR pair mining baseline and our manual assessment used in this RQ.

RAR pair mining baseline. In this RQ, we include the state-of-the-art RAR pair mining technique SinkFinder as baseline by applying both SinkFinder and MIROK on the 20,000 Maven libraries to generate RAR pairs. SinkFinder first mines frequent API pairs from the given code corpus, based on which it further trains API embeddings to infer reliable API pairs and learns to classify RAR pairs. For a fair comparison, we re-implement SinkFinder for Java and apply it to the same code corpus as MIROK.

More specifically, we first mine frequent API pairs (associated with the 20,000 Maven libraries) from the code corpus based on the control flows and data dependencies, following the original mining process of SinkFinder. This results in a total of 50,397 frequent API pairs are obtained as potential RAR pairs, such as <*org.slf4j.Logger.info*, *org.slf4j.Logger.error*> (occuring 9,971 times). Subsequently, we utilize random walks on control-flow graphs to extract API sequences and apply Word2Vec [34] and fastText [9] to train API embeddings, thereby aligning with the approach used in SinkFinder. Building upon the embeddings, a set of seed pairs is employed to iteratively infer reliable positive and negative API pairs, and a binary classification model is trained to identify RAR pairs from the 50,397 frequent API pairs. The seed RAR pairs for initializing the inference process are obtained by matching the 50,397 frequent API pairs with the 26 seed Abs-RAR pairs, in the same matching method presented in Section 3.3. Only one specific seed pair, namely <*ODatabaseDocumentTx.open()*, *ODatabaseDocumentTx.close()*> from the *orientdb-core* library, emerges from this matching process. It's worth noting that having only one seed does not does not pose a threat to the validity of the re-implementation, as SinkFinder is originally crafted to address scenarios where just one seed is available.

Manual Assessment. We manually evaluate the quality of the instantiated RAR pairs. Given the large number of concrete RAR pairs (i.e., MIROK instantiates 6,314 concrete RAR pairs in total), we first use a statistical sampling method [41] to randomly sample 372 concrete RAR pairs, which ensures the estimated precision is in 0.05 error margin at 95% confidence level. Then we invite two developers with more than four years Java development experience to independently inspect the 372 sampled pairs. To make a proper decision, the annotators can search various resources such as library documentation and source code on GitHub to confirm whether the API method pairs are actually for resource acquisition and release. If their annotations for an instantiated RAR pair are inconsistent, a third annotator is assigned to give an additional annotation to resolve the conflict by a majority-win strategy. The Cohen's Kappa agreement [33] of the two annotators is 0.917, indicating substantial agreement.

5.2.2 Results. In total, MIROK instantiates 1,197 valid Abs-RAR pairs into 6,314 RAR pairs from 2,261 libraries. Among the 372 sampled instantiated RAR pairs, 93.3% (347) are confirmed to be valid. In

contrast, the baseline SinkFinder demonstrates an inability to identify any RAR pair when tested on the same extensive code corpus as MiROK. Based on the provided seed, SinkFinder infers only one reliable positive API pair <*ODatabaseDocumentTx.create()*, *ODatabaseDocumentTx.close()*>, causing it fails to perform the iterative reliable pair inference and the subsequent API pair classification. The main reason is that SinkFinder relies on a set of closely-related APIs to train high-quality API embeddings; however, when the code corpus is large, there are too many APIs from different libraries, whose relevance is rather loose, resulting in much less effective API embeddings. Our experimental results show that SinkFinder is not applicable when mining RAR pairs from a large code corpus of different libraries.

Summary: MiROK instantiates 1,197 valid Abs-RAR pairs into 6,314 RAR pairs in 2,261 libraries, and 93.3% of them are valid.

5.3 RQ3.a: Resource Leaks in Online Code Examples

As reported by existing work [12, 54], a non-negligible (e.g., 30%) portion of online code examples (e.g., code snippets in the accepted answers of Stack Overflow posts) may contain defects, such as security issues or resource leak issues, which would further be widely reused by developers in other projects. Therefore, in this RQ, we incorporate our mined Abs-RAR pairs into a light resource leak detection analysis approach to detect resource leaks in online code examples from Stack Overflow posts.

5.3.1 Protocol. We introduce the resource leak detection analysis baseline, the online code example dataset, and evaluation procedure used in this RQ.

Benchmark. We construct a online code example benchmark which contains 46,389 Java code snippets in Stack Overflow. In particular, we first include all the posts tagged with "<java>" from the Stack Overflow dump [1]; then we extract the code snippets surrounded with the tag pair "<pre><code>" and "</code></pre>" from their accepted answers; lastly, we filter out those low-quality code examples with syntactic errors. In this way, we finally collect a benchmark of 46,389 online code examples.

Resource leak analysis baseline. Existing resource leak analysis tools mainly work at byte-code level and are only applicable for a complete compilable project. However, online code examples are often short code snippets without a complete project-level context, and thus existing resource leak analysis tools cannot be directly applied to detect resource leaks in online code examples. Therefore, in this RQ, we first implement a lightweight resource leak analysis approach, which takes Abs-RAR pairs as input and parses code snippets to detect resource leaks at source code level. In particular, for a given code snippet, the lightweight detector first extracts a method call sequence S in the same way as Section 3.1 and transforms all method calls into the form [*O VB NP REST*] in the same way as Section 3.2.1; then for each Abs-RAR pair $\langle res, acq / rel \rangle$, the method sequence S could be considered as containing a resource leak, if (1) a method call $o.m_1$ in S matches $res.acq$, and (2) these is no another method call $o.m_2$ in S that appears after $o.m_1$ and matches $res.rel$. We further compare the resource leaks reported by the lightweight detector when it is incorporated with the 26 basic seed Abs-RAR pairs or with our mined 1,197 valid Abs-RAR pairs.

```
ZipFile zip = new ZipFile(new File( dirToProcess + curFile));
Enumeration<? extends ZipEntry> entries = zip.entries();
while (entries.hasMoreElements()){
    ZipEntry entry = entries.nextElement();
    InputStream xmlStream = zip.getInputStream(entry);
    saxParser.parse( xmlStream, handler );
    xmlStream.close();
}
```

(a) Resource Leak for Abs-RAR Pair ⟨zip, <init> / close⟩

```
Socket socket = new Socket(host, port);
BufferedOutputStream outStream = new BufferedOutputStream(
BufferedInputStream inStream = new BufferedInputStream(new
byte[] buffer = new byte[4096];
for (int read = inStream.read(buffer); read >= 0; read = i
    outStream.write(buffer, 0, read);
inStream.close();
outStream.close();
```

(b) Resource Leak for Abs-RAR Pair ⟨socket, <init> / close⟩

Figure 5: Examples of Valid Resource Leaks

Evaluation procedure. We manually evaluate the reported resource leaks in online code examples. In particular, we first randomly select 256 cases out of all the 761 detected resource leaks, and the sample size 256 is calculated based on the statistical sampling method [41], which ensures the estimated precision is in 0.05 error margin at 95% confidence level. Two developers who have more than four years' Java development experience independently examine the 256 resource leaks and annotate whether the they are true or not. The annotators could search various technical documents and source code on Google and GitHub to understand the given code examples as well as the reported resource leak defects. If their annotations for a case are inconsistent, a third annotator is assigned to give an additional annotation to resolve the conflict by a majority-win strategy. The Cohen's Kappa agreement [33] of the two annotators is 0.929 in our evaluation procedure, indicating substantial agreement.

5.3.2 Results. The detector reports 4.5× more resource leaks with our mined Abs-RAR pairs compared to with the basic seed Abs-RAR pairs. In particular, the detector with our mined Abs-RAR pairs reports 761 resource leaks, while only reports 168 resource leaks with the basic seed Abs-RAR pairs. Among the 256 sampled resource leaks reported by the detector with our Abs-RAR pairs , 73.4% (188) are manually checked as true.

Figure 5 shows two examples of the valid resource leak defects. In the two examples, the resource "zip" and "socket" are acquired by the constructors but not released using the "close" operations. To detect these resource leak defects, we can analyze the resource operations in the code at the conceptual level and do not require knowledge about the corresponding APIs (*e.g.,* "ZipFile" and "Socket") or libraries.

We further analyze the incorrect resource leak defects that are reported by our approach and find that most of them are caused by the inherent limitations in source-code-level analysis (such as

the ambiguous identifiers in code snippets) or the lack of library-specific knowledge. In fact, it is hard to address these limitations for online code examples, as they are often source code snippets without global project contexts.

Summary: With our Abs-RAR pairs, even a simplistic resource leak analysis approach could identify 761 resource leaks in the 46,389 online code examples, and 73.4% of them are manually checked as true resource leaks. The detector reports 4.5X more resource leaks with our Abs-RAR pairs compared to with the basic seed Abs-RAR pairs.

5.4 RQ3.b: Resource Leaks in Open-source Projects

In this RQ, we incorporate our generated RAR pairs with existing resource leak detection tool to investigate whether it could detect more resource leaks in open-source projects.

5.4.1 Protocol. We introduce the resource leak detection analysis baseline, the open-source project benchmark, and evaluation procedure used in this RQ.

Benchmark. We construct a benchmark of 10 open-source projects from GitHub. In particular, we first select 5,000 Java project repositories with more than 20 stars from Github, then filter out the projects that cannot be successfully compiled with "pom.xml" or do not contain any relevant API method of our RAR pairs. To ensure the diversity of projects, we then randomly select 10 projects from the remaining projects. The detailed information of our projects could be found in our replication package [6].

Resource leak analysis baseline. In this RQ, we select the state-of-the-art resource leak detection tool FindBugs [2] as the analysis baseline, as it is a representative static analysis tool that could support general resource leak detection and has been widely used in previous work [44]. In particular, the original FindBugs includes a predefined RAR pair pool which contains stream related RAR pairs. We enhance the original FindBugs by further extending its original RAR pool with our 6,314 newly generated RAR pairs. We denote the enhanced FindBugs as FindBugs*. We apply the original FindBugs and the enhanced FindBugs* to scan projects in the benchmark, respectively.

Evaluation procedure. We manually check the validity of all the resource leaks reported by the original FindBugs and FindBugs*. In particular, two developers with more than four years Java development experience independently examine the reported resource leaks. A third annotator is assigned to resolve the conflict cases by a majority-win strategy.

5.4.2 Results. The original FindBugs reports 9 resource leaks with 4 of them being true resource leaks (i.e., 44.4%); with our RAR pairs, FindBugs* reports 15 resource leaks with 7 of them being true resource leaks (i.e., 46.7%). We further report 3 newly-detected unknown defects to the developers. By the submission time, one of them have been confirmed by the developers. Figure 6 shows the confirmed defect[1] that is newly detected based on our unique RAR pair <*NetClient.connect()*, *NetClient.close()*> of library *vertx-core*.

[1]https://github.com/folio-org/okapi/pull/1303

```
NetClient c = vertx.createNetClient(options);
return c.connect(port, "localhost").compose(
    socket -> socket.close()
        .otherwiseEmpty().compose(x -> {
          if (iter == 0) {
            return Future.failedFuture(messages.getMessage("11503", Integer
          }
          Promise<Void> promise = Promise.promise();
          vertx.setTimer(100, id -> waitPortToClose(iter - 1).onComplete(pr
          return promise.future();
        }),
    noSocket -> Future.succeededFuture());
```

Figure 6: The Confirmed Resource Leak Defect

As shown in the figure, the resource leak occurs, as the *NetClient* resource *c* is acquired (in line *c.connect(...)*) but without subsequently calling the release method *c.close(...)*. The original FindBugs fails to detect this resource leak, as its initial RAR pair pool does not contain the RAR pair <*NetClient.connect()*, *NetClient.close()*> and it is unaware of *NetClient* being a resource object.

Summary: Our new RAR pairs help existing resource leak detection tool FindBugs report more unknown defects without reducing its precision. Our results indicate that our mined RAR pairs could be incorporated into existing resource leak detection to enable more powerful resource leak detection.

5.5 RQ4: Impact of Each Mining Strategy

In this RQ, we investigate the contribution of each mining strategy (i.e., rule-based Abs-RAR pair expansion and learning-based Abs-RAR pair identification).

5.5.1 Protocol. We compare the effectiveness of the following variants of MIROK to study the contribution of each mining strategy: (1) *MIROK-L*, which removes learning-based Abs-RAR pair identification and only includes rule-based Abs-RAR pair expansion; and (2) *MIROK-R*, which removes rule-based Abs-RAR pair expansion and only includes learning-based Abs-RAR pair identification.

5.5.2 Results. The results of the impact of the two mining strategies are shown in Figure 7, which provides the numbers of the mined Abs-RAR pairs in different iterations. We could observe that MIROK mines much more Abs-RAR pairs with two strategies together. With the learning-based Abs-RAR pair identification removed, *MIROK-L* only mines a few Abs-RAR pairs in the first iteration and cannot mine more abstract Abs-RAR pairs in the subsequent iterations. With the rule-based Abs-RAR pair expansion removed, *MIROK-R* only mines 38.0% fewer Abs-RAR pairs than the complete MIROK. These results suggest that both the two strategies are important for MIROK, and both strategies are complementary in their capabilities of Abs-RAR pair mining. In particular, rule-based Abs-RAR pair expansion can derive high-quality Abs-RAR pairs from existing ones based on simple rules and plays an important role in early iterations, while learning-based Abs-RAR pair identification can continuously identify Abs-RAR pairs in a broader scope.

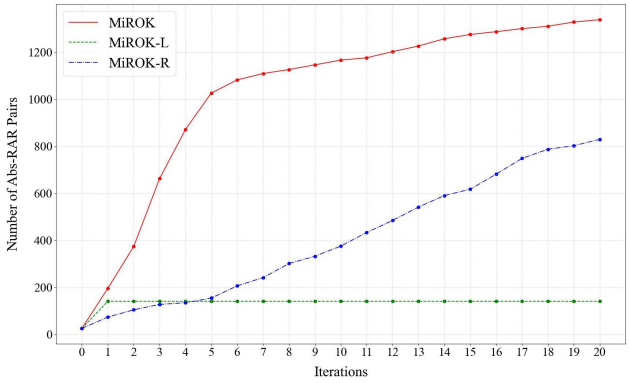

Figure 7: Numbers of Mined Abs-RAR Pairs with Different MiROK Variants

Summary: Both the rule-based Abs-RAR pair expansion and the learning-based Abs-RAR pair identification are important for MiROK. These two strategies are complementary in their capabilities of Abs-RAR pair mining.

6 THREATS TO VALIDITY

The threats to the internal validity of our studies lie in the randomness of data sampling and the subjectiveness in data annotation. To mitigate these threats, we follow commonly-used data sampling strategy by controlling the estimated precision within 0.05 error margin at 95% confidence level, and multiple annotators are involved with high agreement coefficients. Thee threats to the external validity lies in the benchmarks used by our work, which cannot guarantee the generality of our findings. To minimize such threats, we leverage a large scale of code corpus and libraries and include two resource leak detection scenarios for evaluation. We believe it is interesting future work to extend MiROK to other programming languages and incorporating MiROK with more resource leak analysis tools.

7 CONCLUSIONS AND FUTURE WORK

In this work, we propose MiROK, a novel mining approach which represents resource-operation knowledge as abstract resource acquisition/release operation pairs (Abs-RAR pairs), and mine such Abs-RAR pairs from a large code corpus. Given a large code corpus, MiROK first mines Abs-RAR pairs with novel rule-based pair expansion and learning-based pair identification strategies, and then instantiates these Abs-RAR pairs into concrete RAR pairs. We implement MiROK and apply it to mine RAR pairs from a large code corpus of 1,454,224 Java methods and 20,000 Maven libraries. We then perform an extensive evaluation to investigate the mining effectiveness of MiROK and the practical usage of its mined RAR pairs for supporting resource leak detection. Our results show that MiROK mines 1,313 new Abs-RAR pairs and instantiates them into 6,314 RAR pairs with a high precision (i.e., 93.3%). In addition, we feed our mined RAR pairs to existing resource leak analysis approaches, and help them detect more resource leak bugs in both online code examples and open-source projects. Our results indicate both the high quality and practical usage of our mined RAR pairs. In addition, we further perform an ablation study to show the contribution of each mining strategy in MiROK.

8 DATA AVAILABILITY

Our replication package is at [6].

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grant 61972098.

REFERENCES

[1] 2021. *Stack Overflow data dump version from March 4, 2021.* Retrieved September 4, 2021 from https://archive.org/download/stackexchange/
[2] 2022. *FindBugs.* Retrieved Augest 5, 2022 from http://findbugs.sourceforge.net//
[3] 2022. *javalang.* Retrieved August 5, 2022 from https://github.com/c2nes/javalang
[4] 2022. *Libraries.io.* Retrieved August 5, 2022 from https://libraries.io/maven
[5] 2022. *PyTorch.* Retrieved August 5, 2022 from https://pytorch.org/
[6] 2022. *Replication Package.* Retrieved August 5, 2022 from https://mirok-replication.github.io/
[7] Mithun Acharya, Tao Xie, Jian Pei, and Jun Xu. 2007. Mining API patterns as partial orders from source code: from usage scenarios to specifications. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007*, Ivica Crnkovic and Antonia Bertolino (Eds.). ACM, 25–34. https://doi.org/10.1145/1287624.1287630
[8] Pan Bian, Bin Liang, Jianjun Huang, Wenchang Shi, Xidong Wang, and Jian Zhang. 2020. SinkFinder: harvesting hundreds of unknown interesting function pairs with just one seed. In *Proceedings of 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1101–1113. https://doi.org/10.1145/3368089.3409678
[9] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2017. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguistics* 5 (2017), 135–146. https://doi.org/10.1162/tacl_a_00051
[10] Ray-Yaung Chang, Andy Podgurski, and Jiong Yang. 2007. Finding what's not there: a new approach to revealing neglected conditions in software. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2007, London, UK, July 9-12, 2007*, David S. Rosenblum and Sebastian G. Elbaum (Eds.). ACM, 163–173. https://doi.org/10.1145/1273463.1273486
[11] Navid Emamdoost, Qiushi Wu, Kangjie Lu, and Stephen McCamant. 2021. Detecting Kernel Memory Leaks in Specialized Modules with Ownership Reasoning. In *Proceedings of 28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021.* The Internet Society. https://www.ndss-symposium.org/ndss-paper/detecting-kernel-memory-leaks-in-specialized-modules-with-ownership-reasoning/
[12] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. 2017. Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security. In *Proceedings of 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017.* IEEE Computer Society, 121–136. https://doi.org/10.1109/SP.2017.31
[13] Mohammadreza Ghanavati, Diego Costa, Janos Seboek, David Lo, and Artur Andrzejak. 2020. Memory and resource leak defects and their repairs in Java projects. *Empir. Softw. Eng.* 25, 1 (2020), 678–718. https://doi.org/10.1007/s10664-019-09731-8
[14] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 933–944. https://doi.org/10.1145/3180155.3180167
[15] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 631–642. https://doi.org/10.1145/2950290.2950334
[16] Chaorong Guo, Jian Zhang, Jun Yan, Zhiqiang Zhang, and Yanli Zhang. 2013. Characterizing and detecting resource leaks in Android applications. In *Proceedings of 2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, Ewen Denney, Tevfik Bultan, and Andreas Zeller (Eds.). IEEE, 389–398. https://doi.org/10.1109/ASE.2013.6693097
[17] Samir Gupta, Sana Malik, Lori L. Pollock, and K. Vijay-Shanker. 2013. Part-of-speech tagging of program identifiers for improved text-based software engineering tools. In *Proceedings of IEEE 21st International Conference on Program*

Comprehension, ICPC 2013, San Francisco, CA, USA, 20-21 May, 2013. IEEE Computer Society, 3–12. https://doi.org/10.1109/ICPC.2013.6613828

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[19] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. *CoRR* abs/1909.09436 (2019). arXiv:1909.09436 http://arxiv.org/abs/1909.09436

[20] Dan Jurafsky. 2000. *Speech & language processing*. Pearson Education India.

[21] Martin Kellogg, Narges Shadab, Manu Sridharan, and Michael D. Ernst. 2021. Lightweight and modular resource leak verification. In *Proceedings of 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 181–192. https://doi.org/10.1145/3468264.3468576

[22] Martin Kellogg, Narges Shadab, Manu Sridharan, and Michael D. Ernst. 2021. Lightweight and modular resource leak verification. In *Proceedings of 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 181–192. https://doi.org/10.1145/3468264.3468576

[23] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[24] Keshav Kolluru, Vaibhav Adlakha, Samarth Aggarwal, Mausam, and Soumen Chakrabarti. 2020. OpenIE6: Iterative Grid Labeling and Coordination Analysis for Open Information Extraction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 3748–3761. https://doi.org/10.18653/v1/2020.emnlp-main.306

[25] Sangho Lee, Changhee Jung, and Santosh Pande. 2014. Detecting memory leaks through introspective dynamic behavior modelling using machine learning. In *Proceedings of 36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, Pankaj Jalote, Lionel C. Briand, and André van der Hoek (Eds.). ACM, 814–824. https://doi.org/10.1145/2568225.2568307

[26] Mengchen Li, Yuanjun Chen, Linzhang Wang, and Guoqing Xu. 2013. Dynamically validating static memory leak warnings. In *Proceedings of International Symposium on Software Testing and Analysis, ISSTA '13, Lugano, Switzerland, July 15-20, 2013*, Mauro Pezzè and Mark Harman (Eds.). ACM, 112–122. https://doi.org/10.1145/2483760.2483778

[27] Wen Li, Haipeng Cai, Yulei Sui, and David Manz. 2020. PCA: memory leak detection using partial call-path analysis. In *Proceedings of 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1621–1625. https://doi.org/10.1145/3368089.3417923

[28] Zhenmin Li and Yuanyuan Zhou. 2005. PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2005, Lisbon, Portugal, September 5-9, 2005*, Michel Wermelinger and Harald C. Gall (Eds.). ACM, 306–315. https://doi.org/10.1145/1081706.1081755

[29] Yepang Liu, Jue Wang, Lili Wei, Chang Xu, Shing-Chi Cheung, Tianyong Wu, Jun Yan, and Jian Zhang. 2019. DroidLeaks: a comprehensive database of resource leaks in Android apps. *Empir. Softw. Eng.* 24, 6 (2019), 3435–3483. https://doi.org/10.1007/s10664-019-09715-8

[30] Yepang Liu, Chang Xu, Shing-Chi Cheung, and Valerio Terragni. 2016. Understanding and detecting wake lock misuses for Android applications. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su (Eds.). ACM, 396–409. https://doi.org/10.1145/2950290.2950297

[31] Yi Liu, Yadong Yan, Chaofeng Sha, Xin Peng, Bihuan Chen, and Chong Wang. 2022. DeepAnna: Deep Learning based Java Annotation Recommendation and Misuse Detection. In *Proceedings of IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, Honolulu, HI, USA, March 15-18, 2022*. IEEE, 685–696. https://doi.org/10.1109/SANER53432.2022.00086

[32] Jun Ma, Sheng Liu, Shengtao Yue, Xianping Tao, and Jian Lu. 2017. LeakDAF: An Automated Tool for Detecting Leaked Activities and Fragments of Android Applications. In *Proceedings of 41st IEEE Annual Computer Software and Applications Conference, COMPSAC 2017, Turin, Italy, July 4-8, 2017. Volume 1*, Sorel Reisman, Sheikh Iqbal Ahamed, Claudio Demartini, Thomas M. Conte, Ling Liu, William R. Claycomb, Motonori Nakamura, Edmundo Tovar, Stelvio Cimato, Chung-Horng Lung, Hiroki Takakura, Ji-Jiang Yang, Toyokazu Akiyama, Zhiyong Zhang, and Kamrul Hasan (Eds.). IEEE Computer Society, 23–32. https://doi.org/10.1109/COMPSAC.2017.161

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

[34] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. (2013), 3111–3119. https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

[35] Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNLP, 2-7 August 2009, Singapore*, Keh-Yih Su, Jian Su, and Janyce Wiebe (Eds.). The Association for Computer Linguistics, 1003–1011. https://aclanthology.org/P09-1113/

[36] Hoan Anh Nguyen, Robert Dyer, Tien N. Nguyen, and Hridesh Rajan. 2014. Mining preconditions of APIs in large-scale code corpus. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne D. Storey (Eds.). ACM, 166–177. https://doi.org/10.1145/2635868.2635924

[37] Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H. Pham, Jafar M. Al-Kofahi, and Tien N. Nguyen. 2009. Graph-based mining of multiple object usage patterns. In *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009, Amsterdam, The Netherlands, August 24-28, 2009*, Hans van Vliet and Valérie Issarny (Eds.). ACM, 383–392. https://doi.org/10.1145/1595696.1595767

[38] Sebastian Nielebock, Robert Heumüller, Kevin Michael Schott, and Frank Ortmeier. 2021. Guided pattern mining for API misuse detection by change-based code analysis. *Autom. Softw. Eng.* 28, 2 (2021), 15. https://doi.org/10.1007/s10515-021-00294-x

[39] Jacques A. Pienaar and Robert Hundt. 2013. JSWhiz: Static analysis for JavaScript memory leaks. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2013, Shenzhen, China, February 23-27, 2013*. IEEE Computer Society, 11:1–11:11. https://doi.org/10.1109/CGO.2013.6495007

[40] Suman Saha, Jean-Pierre Lozi, Gaël Thomas, Julia L. Lawall, and Gilles Muller. 2013. Hector: Detecting Resource-Release Omission Faults in error-handling code for systems software. In *Proceedings of 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, June 24-27, 2013*. IEEE Computer Society, 1–12. https://doi.org/10.1109/DSN.2013.6575307

[41] Ravindra Singh and Naurang Singh Mangat. 2013. *Elements of Survey Sampling*. Vol. 15. Springer Science & Business Media.

[42] Yulei Sui, Ding Ye, and Jingling Xue. 2012. Static memory leak detection using full-sparse value-flow analysis. In *Proceedings of International Symposium on Software Testing and Analysis, ISSTA 2012, Minneapolis, MN, USA, July 15-20, 2012*, Mats Per Erik Heimdahl and Zhendong Su (Eds.). ACM, 254–264. https://doi.org/10.1145/2338965.2336784

[43] Yulei Sui, Ding Ye, and Jingling Xue. 2014. Detecting Memory Leaks Statically with Full-Sparse Value-Flow Analysis. *IEEE Trans. Software Eng.* 40, 2 (2014), 107–122. https://doi.org/10.1109/TSE.2014.2302311

[44] Emina Torlak and Satish Chandra. 2010. Effective interprocedural resource leak detection. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel (Eds.). ACM, 535–544. https://doi.org/10.1145/1806799.1806876

[45] Chong Wang, Xin Peng, Mingwei Liu, Zhenchang Xing, Xuefang Bai, Bing Xie, and Tuo Wang. 2019. A learning-based approach for automatic construction of domain glossary from source code and documentation. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*, Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo (Eds.). ACM, 97–108. https://doi.org/10.1145/3338906.3338963

[46] Chong Wang, Xin Peng, Zhenchang Xing, and Xiujie Meng. 2023. Beyond Literal Meaning: Uncover and Explain Implicit Knowledge in Code Through Wikipedia-Based Concept Linking. *IEEE Trans. Software Eng.* 49, 5 (2023), 3226–3240. https://doi.org/10.1109/TSE.2023.3250029

[47] Chong Wang, Xin Peng, Zhenchang Xing, Yue Zhang, Mingwei Liu, Rong Luo, and Xiujie Meng. 2023. XCoS: Explainable Code Search Based on Query Scoping and Knowledge Graph. *ACM Trans. Softw. Eng. Methodol.* (apr 2023). https://doi.org/10.1145/3593800 Just Accepted.

[48] Qian Wu, Guangtai Liang, Qianxiang Wang, Tao Xie, and Hong Mei. 2011. Iterative mining of resource-releasing specifications. In *Proceedings of 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*, Perry Alexander, Corina S. Pasareanu, and John G. Hosking (Eds.). IEEE Computer Society, 233–242. https://doi.org/10.1109/ASE.2011.6100058

[49] Tianyong Wu, Jierui Liu, Zhenbo Xu, Chaorong Guo, Yanli Zhang, Jun Yan, and Jian Zhang. 2016. Light-Weight, Inter-Procedural and Callback-Aware Resource Leak Detection for Android Apps. *IEEE Trans. Software Eng.* 42, 11 (2016), 1054–1076. https://doi.org/10.1109/TSE.2016.2547385

[50] Dacong Yan, Shengqian Yang, and Atanas Rountev. 2013. Systematic testing for resource leaks in Android applications. In *Proceedings of IEEE 24th International Symposium on Software Reliability Engineering, ISSRE 2013, Pasadena, CA, USA, November 4-7, 2013*. IEEE Computer Society, 411–420. https://doi.org/10.1109/ISSRE.2013.6698894

[51] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Zi Qun Ang, Jing Li, and Nachiket Kapre. 2016. Software-Specific Named Entity Recognition in Software Engineering Social Content. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - Volume 1*. IEEE Computer Society, 90–101. https://doi.org/10.1109/SANER.2016.10

[52] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Jing Li, and Nachiket Kapre. 2016. Learning to Extract API Mentions from Informal Natural Language Discussions. In *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*. IEEE Computer Society, 389–399. https://doi.org/10.1109/ICSME.2016.11

[53] Insu Yun, Changwoo Min, Xujie Si, Yeongjin Jang, Taesoo Kim, and Mayur Naik. 2016. APISan: Sanitizing API Usages through Semantic Cross-Checking. In *Proceedings of 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 363–378. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/yun

[54] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. 2018. Are code examples on an online Q&A forum reliable?: a study of API misuse on stack overflow. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 886–896. https://doi.org/10.1145/3180155.3180260

[55] Hao Zhong, Tao Xie, Lu Zhang, Jian Pei, and Hong Mei. 2009. MAPO: Mining and Recommending API Usage Patterns. In *Proceedings of ECOOP 2009 - Object-Oriented Programming, 23rd European Conference, Genoa, Italy, July 6-10, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5653)*, Sophia Drossopoulou (Ed.). Springer, 318–343. https://doi.org/10.1007/978-3-642-03013-0_15

Received 2023-02-02; accepted 2023-07-27