Security Tools' API Recommendation Using Machine Learning

Zarrin Tasnim Sworna[2,3], Anjitha Sreekumar[1,2], Chadni Islam[1,2] and Muhammad Ali Babar[1,2,3]
[1]*Centre for Research on Engineering Software Technologies (CREST), University of Adelaide, Australia*
[2]*School of Computer Science, University of Adelaide, Australia*
[3]*Cyber Security Cooperative Research Centre, Australia*

Keywords: Security Tools' API, Security Orchestration, API Recommendation, Security Operation Center.

Abstract: Security Operation Center (SOC) teams manually analyze numerous tools' API documentation to find appropriate APIs to define, update and execute incident response plans for responding to security incidents. Manually identifying security tools' APIs is time consuming that can slow down security incident response. To mitigate this manual process's negative effects, automated API recommendation support is desired. The state-of-the-art automated security tool API recommendation uses Deep Learning (DL) model. However, DL models are environmentally unfriendly and prohibitively expensive requiring huge time and resources (denoted as "Red AI"). Hence, "Green AI" considering both efficiency and effectiveness is encouraged. Given SOCs' incident response is hindered by cost, time and resource constraints, we assert that Machine Learning (ML) models are likely to be more suitable for recommending suitable APIs with fewer resources. Hence, we investigate ML model's applicability for effective and efficient security tools' API recommendation. We used 7 real world security tools' API documentation, 5 ML models, 5 feature representations and 19 augmentation techniques. Our Logistic Regression model with word and character level features compared to the state-of-the-art DL-based approach reduces 95.91% CPU core hours, 97.65% model size, 291.50% time and achieves 0.38% better accuracy, which provides cost-cutting opportunities for industrial SOC adoption.

1 INTRODUCTION

The frequency of security incidents increased by 358% from 2019 to 2020 (Instinct, 2021). Security Operation Centers (SOCs) use Incident Response Plan (IRP) to respond to the security incidents. IRP is a sequence of tasks that are performed by orchestrating various security tools in a Security Orchestration, Automation and Response (SOAR) platform in response to a specific security incident (PAN, 2019). Table 1 shows an example IRP for malware investigation from the Cortex SOAR platform (PAN, 2022). It presents some tasks of an IRP and the required APIs from different security tools for executing these tasks. An organization uses 76 security tools on average (Muncaster, 2021). Each of these tools has a large number of APIs. For instance, Splunk (Splunk, 2022) and Malware Information Sharing Platform, MISP (MISP, 2022) have around 598 and 398 APIs, respectively. To define, update or execute an IRP, SOAR developers and SOC teams need to manually search and select the APIs from diverse security tools (Sworna et al., 2022). For example, the Phantom SOAR platform requires around 1900 APIs for executing IRPs using diverse tools (Phantom, 2021). Hence, a SOC team with time constraint (PAN, 2019) finds manually reading documentation to find APIs of these numerous tools as hard to navigate, cumbersome and time consuming (Robillard and DeLine, 2011). Such manual efforts for incident response cause a significant human burden and contribute to fatigue in SOCs (Vielberth et al., 2020). To accelerate incident response and reduce SOC teams' burden, there is a dire need for automated API recommendation support (Sworna et al., 2022).

To support automated programming language-specific (e.g., Java) API recommendation, several Deep Learning (DL)-based approaches have been reported in Software Engineering (SE) literature (Ling et al., 2020), (Gu et al., 2016). Besides, a set of SE studies (Cai et al., 2019), (Huang et al., 2018), (Nguyen et al., 2017), (Ye et al., 2016) used similarity score (SimScore) (Ye et al., 2016) on DL-based word embedding for language-specific API recommendation. The state-of-the-art security tool API recommendation called APIRO (Sworna et al., 2022) presents a DL-based (i.e., Convolutional Neural Network (CNN)) approach, which outperformed Recurrent Neural Network (RNN)-based DL approach.

Though DL models achieve high accuracy, they are computationally quite expensive due to the requirement of complex hyper-parameter and architecture tuning, millions and billions of weights and connections between units and powerful hardware resources (LeCun et al., 2015). The demand for computing resources to train DL models has increased 300,000-fold from 2012 to 2019 (Strubell et al., 2019). Moreover, DL has a black-box nature and running DL even using powerful hardware and algorithmic parallelization still requires hours to weeks of long training time (Fu and Menzies, 2017). For example, Deep-API (Gu et al., 2016) required 240 hours of GPU time for training DL-based API usage recommendation.

Furthermore, DL models are environmentally unfriendly and prohibitively expensive, which is denoted as “Red AI” (Strubell et al., 2019). The necessary computation power for training a single DL model emitted 626,000 tonnes CO_2, which is five times more than an average car emits throughout its lifetime (Strubell et al., 2019). As organizations are adopting cloud environments, the total financial cost of using a DL model in SOC can be expensive due to the highly monetized cloud environments (Fu and Menzies, 2017). Hence, “Green AI” (Strubell et al., 2019) is encouraged, which considers efficiency as the primary assessment criterion along with effectiveness. However, the existing DL-based SE studies (Choetkiertikul et al., 2018), (Mou et al., 2016), (White et al., 2015) did not focus on computation efficiency (i.e., resource usage).

An increasing number of organizations prefer sustainability-based designs (i.e., use fewer resources for acquiring outcome) in industrial SE solutions for extensive cost-cutting opportunities (Calero and Piattini, 2015). If two competing learning models produce similar results, the preferred model is the simpler one to understand and interpret (Fakhoury et al., 2018). Given SOCs operate under critical cost, time and resource constraints (CSCRC, 2022), we assert that traditional ML-based, compared with DL-based, solution can be more attractive for SOCs as traditional ML models are usually simpler, which require less time, resource and cost (Fakhoury et al., 2018). However, the performance of ML and DL varies based on various contexts such as domain, downstream task and dataset. For example, in the SE domain, DL outperformed ML in opinion-based question-answer extraction (Chatterjee et al., 2021). In contrast, ML outperformed DL in finding similar questions in Stack Overflow (Fu and Menzies, 2017). To support a SOC team by providing a time and resource efficient and effective security tool API recommender, there is a gap for an investigation of the applicability of ML over DL.

Table 1: An example IRP for malware investigation with required security tools’ APIs for executing the IRP.

IRP Tasks	Security Tool	Security Tool’s API
Send file to Cuckoo	Cuckoo	Post/tasks/create/file
Get task report	Cuckoo	Get/tasks/report/id/format
Delete malicious file	Wazuh	delete/manager/files
Block IP/URL IOCs	Zeek	NetControl::drop_address
Isolate endpoint	Zeek	NetControl:: quarantine_host

To address the above-mentioned gap, we empirically investigate the applicability of traditional ML models for recommending security tools’ API with significant cost-cutting opportunities. We use the API documentation of 7 diverse real world security tools and adopt 19 text augmentation techniques for semantic variation enrichment. We compare and identify the best performing ML model out of 5 popular ML models (e.g., Logistic Regression (LR)). We explore 5 feature (e.g., word and character level) representations techniques for each ML model. Later, we compare the effectiveness and efficiency of the best performing ML model with the baselines that are SimScore and state-of-the-art DL-based APIRO.

Our experimental results show that our best performing ML model for security tool API recommendation is the LR model with word and character level features as it outperformed the other ML models and baselines both in effectiveness (e.g., accuracy) and efficiency (e.g., resource). Our best performing ML model reduces the required memory (i.e., GB) in terms of 97.65% model size, 99.13% maximum disk write, 92.86% maximum disk read and reduces 95.91% CPU core hours compared to APIRO. It also gains 0.38% better accuracy with 291.50% less training time compared to APIRO. Hence, our best performing ML model recommends the correct APIs requiring less time and resources that achieves sustainable design goals compared to the DL-based approach for fast incident response in real world SOCs.

The main contributions of this study are:

1. We empirically investigate various simple ML models and features to choose the best performing ML model for recommending security tool APIs.
2. We compare our best performing ML model’s effectiveness with SimScore and DL-based APIRO, where our best performing ML model outperformed the baselines in terms of accuracy, mean reciprocal rank and mean precision@K.
3. We highlight efficiency showing our best performing ML model reduces 95.91% and 97.65% of CPU core hours and model size, respectively with 291.50% less training time compared to APIRO.

Table 2: Security tools' diversity based on the type of tool, API and API data source.

Tool Name	Tool Type	API	API Data Source
Limacharlie	EDR	REST, Python, CLI Commands	JSON, HTML
Cuckoo	Sandbox	REST	HTML
MISP	TIP	REST, Python	HTML
Zeek	NIDS	Python, CLI Commands	HTML
Snort	NIDS	CLI Commands	HTML
Wazuh	HIDS	REST	HTML
Splunk	SIEM	REST	YAML, HTML

This paper is structured as follows. Our study design and results are detailed in Sections 2 and 3, respectively. Sections 4 and 5 present threats to validity and implications. Section 6 summarises the related work and Section 7 concludes our paper.

2 STUDY DESIGN

This section presents our Research Questions (RQ), our security tool APIs' corpus creation and pre-processing methods, methods to answer RQs and the used evaluation metrics.

2.1 Research Questions

The following RQs motivated our empirical study.
RQ1: To What Extent Are the Traditional ML Models Viable Approaches to Recommend Security Tools' API? This RQ investigates to what extent traditional ML models can perform to recommend security tools' API. We also aim to identify the simple statistical feature representation that helps the ML model to gain better performance. Thus, we seek to identify the best performing traditional ML model and feature. An answer to this RQ will help a SOC team to select the best performing ML model with features in their SOC to recommend security tools' API.
RQ2: How Does Traditional ML Perform Compared to the State-of-the-Art Baselines? In this RQ, we compare our best performing traditional ML model that we gained from RQ1 to the following baselines: (i) APIRO which is the state-of-the-art security tool API recommendation approach using a CNN-based DL model (Sworna et al., 2022) and (ii) classical SimScore approach using a similarity score method on DL-based word embedding that is widely used for Java API recommendation (Cai et al., 2019), (Huang et al., 2018), (Ye et al., 2016). An answer to this RQ will identify whether the traditional ML model can gain similar, worse or better performance than DL baselines. This comparison will help the SOC team to select whether to adopt the ML model over baselines.
RQ3: What Are the Required Time and Resource Utilization for Models to Recommend Security Tools' API? Since time and resource constraints are the significant barriers to SOCs' incident response (CSCRC, 2022), time and resource utilization must be evaluated to validate the usability of the API recommendation support in the real world SOC environment. Delay in API recommendation may cause a delay in the IRP execution in SOC that may result in a significant loss for the organization due to the negative impact of an attack (Vielberth et al., 2020). Besides, to respond to the ever increasing and intricate threat landscape, new security tools and new APIs of the existing tools are constantly being added to the SOC (Muncaster, 2021). To keep the security tool API recommender of SOC up-to-date, re-training the model is a must. Hence, we analyze time (i.e., training, testing) and resource utilization (e.g., model size, CPU core hours) of the best performing ML model compared to baselines to inspect the practical usability for deployment in the real SOC setting. The result of RQ3 will help the SOC team to choose a sustainable, time and resource efficient model for automated API recommendation.

2.2 Security Tool APIs' Corpus Construction and Pre-Processing

In this section, we present the methodology for security tool APIs' corpus construction and pre-processing.

2.2.1 Security Tool APIs' Corpus Construction

We created a corpus of APIs using 7 diverse popular real world security tools. This includes Limacharlie (Limacharlie, 2022), MISP (MISP, 2022), Snort (Snort, 2020), Cuckoo (Cuckoo, 2019), Splunk (Splunk, 2022), Zeek (Zeek, 2020) and Wazuh (Wazuh, 2022). These diverse types of security tools perform varied functionalities to ensure security. These tools also vary in other criteria (e.g., API type and API data source) as reported in Table 2. The API data of different tools help us to evaluate our models' effectiveness for API recommendation in diverse data settings. We selected these tools as a common use case scenario in SOC is detecting endpoint and network attacks using Endpoint Detection & Response (EDR) tool (i.e., **Limacharlie**) and Network Intrusion Detection System (NIDS) (i.e., **Zeek** and **Snort**). Security information and event management (SIEM) tool (i.e., **Splunk**) helps incident investigation and forensic analysis in SOC. Besides, SOC collects updated malware

Table 3: Statistics of API data collected from security tools.

Security Tool	API	API Num	Total Num of Words	Mean Word Count in Description
Lima-charlie	Python	146	2395	8.81
	REST	84		
	Sensor Commands	42		
Cuckoo	REST API	23	685	18.47
	Distributed Cuckoo	12		
	Process	9		
MISP	Automation & MISP	66	5424	14.05
	PyMISP Python Lib	20		
	PyMISP Python	312		
Zeek	Python	63	1571	13.90
	NetControl	31		
	Command-Line	19		
Snort	Snort (Up-to 2.2)	145	1577	10.88
Wazuh	REST	152	2387	15.70
Splunk	REST	598	4065	6.81
Total		1722	17504	10.73

*API num means the number of APIs collected from the API document.

information from Threat Intelligence Platform (TIP) (i.e., **MISP**) and analyses malware using sandbox tools (i.e., **Cuckoo**). For threat detection and response of a particular host, SOC uses Host Intrusion Detection Systems (HIDS) (i.e., **Wazuh**).

Since different security tools provide their API documentation in different formats (e.g., JSON, HTML), we built different scrapers using diverse libraries. We utilized BeautifulSoup (Beautifulsoup, 2020) library for parsing Splunk, Cuckoo and Zeek HTML pages, Python built-in JSON package to parse Limacharlie JSON data and Python YAML module to parse Wazuh YAML pages. Lastly, we created a unified API corpus, API_c, of security tools by gathering the data collected from these seven tools. As shown in Table 3, we collected 1722 APIs and their respective descriptions, parameters and return values. The total word count of the API descriptions is 17k. For each API description, the average word count is 10.73.

2.2.2 Security Tool API Corpus Pre-Processing

We pre-processed the APIs' textual descriptions as data pre-processing contributes to the success of the model in the data science pipeline (Biswas et al., 2021).

Noise and Stop-Word Removal: To clean APIs' textual description, we removed noises to retain the alphanumerical data along with underscore and minus in the description. Underscore was retained to prevent the formation of sub-terms from a single term (e.g. attribute_identifier). Minus was retained to keep argument representations (e.g. -u). We removed stopwords using the NLTK (Steven Bird and Loper, 2009) English stop-word list.

Lowercasing: We performed lowercasing to represent words of different cases (e.g., Cuckoo, cuckoo) to the same lower-case form (e.g., cuckoo).

Lemmatization: We performed WordNet-based lemmatization using NLTK (Steven Bird and Loper, 2009) to represent a word's inflected forms (e.g., getting, gets) to its dictionary-based root form (e.g., get).

API Clustering: We created clusters of APIs of a tool, which differ based on class name, method name, parameters or representations, but have identical API description (Sworna et al., 2022). Clustering helps to provide a comprehensive view of APIs of a tool that performs the same intended task. We performed automated API clustering by exact description matching with a Python script. The clustering created a list of 55 API clusters by merging 143 APIs. Our clustering resulted in the API corpus of 1466 distinctive descriptions with the corresponding APIs.

Text Augmentation: Since natural language queries can have synonyms, para-phrases and spelling mistakes, we enrich the API corpus with diverse text augmentation techniques for semantic variation enrichment. We enriched the API descriptions of API_c corpus using various text augmentation techniques (e.g., synonym substitute using Wordnet, synonym substitute using PPDB) to improve the model's performance. Implementing text augmentation is challenging as substituting some words may change the context and semantics. For instance, synonym substitute using Wordnet substitutes the word 'PID' with 'Pelvic inflammatory disease' that changes the context as 'PID' is a file of Snort tool (Snort, 2020). Hence, we need to build an immutable word list, which will not be changed during augmentation. The immutable words refer to domain-specific words (e.g., malware, HIDS) that are usually Nouns (Sworna et al., 2022). Firstly, we created a list of words under the Noun tag from API_c using NLTK (Steven Bird and Loper, 2009) POS tagger and Universal Part-of-Speech Tagset. Then, we built the immutable word corpus by inspecting that list. Two researchers performed immutable word selection with Cohen's Kappa (McHugh, 2012) of 0.78, which indicates substantial agreement.

For text augmentation, 19 different suitable augmentation techniques were used, which are reported to improve the performance of security tools' API recommendation (Sworna et al., 2022). We implemented these augmentation techniques (listed in our online Appendix [1]) on the API corpus using NLPAug (Ma, 2019) library. Hence, the augmented corpus API_c included the original data and augmented data.

Processed API Corpus: To build the processed API corpus, the augmented API descriptions of the corpus were further processed by removing noises and stopwords, lowercasing and lemmatizing.

[1]https://tinyurl.com/2bjhp5x8

2.3 Research Methods to Answer RQs

This section presents the research methods used for answering our RQs. We ran our experiments in a computing cluster including 10 CPU cores with 10GB RAM of Linux NeXtScale system (Intel X86-64). We augmented data for each API adopting text augmentation techniques from the labeled data of API documentation as API documentation provides API with relevant description representing labeled data. Thus, the augmented data mitigated the need of time and effort consuming manual labeling of a large number of data for validating the prediction model. We randomly shuffled our corpus and performed stratified 80-20 train-test split, which is commonly used in the literature (Yenigalla et al., 2018), (Sworna et al., 2022). We performed 10-fold cross-validation on that 80% train dataset for hyper-parameter optimization, which is the recommended practice for learning-based models (Scikit-learn, 2022b). We used the 20% test dataset for the models' evaluation. Since queries can have synonyms, para-phrases and contextual similar words, our generalized test query set represents a wider variety of Natural Language (NL) queries as the augmented description ensures NL variation of the API descriptions as queries (Sworna et al., 2022).

RQ1. Traditional ML Models' Performance Investigation: In this RQ, we explore five traditional ML models for security tool API recommendation such as (i) Naïve Bayes (NB); (ii) Support Vector Machine (SVM); (iii) Logistic Regression (LR); (iv) Random Forrest (RF); and (v) eXtreme Gradient Boosting (XGB). These models belong to different categories such as Bayesian networks, Support vector machines, Regression and Ensemble. We consider these models as they are representative of the most investigated models and considered state-of-the-art in SE for documentation classification (Fucci et al., 2019), security text prediction (Le et al., 2020) and SE-specific text prediction and classification (Fu and Menzies, 2017). Besides, the adoption of two to five representative ML models is a usual practice in the literature (Fakhoury et al., 2018), (Fucci et al., 2019).

Our goal is to study how well simple ML, without complex feature engineering or Natural Language Processing (NLP) techniques (e.g., NN-based embeddings), can recommend security tools' API. Hence, we consider simple statistical features that are easy to compute and improve the performance of ML models (Haque et al., 2022). To obtain the relevant features from the API corpus, we used different Term Frequency-Inverse Document Frequency (TF-IDF)-based feature representations. For each ML model, we compared TF-IDF-based word level, word n-gram level, character (char) level and the combination of word and char level features along with NLP/text-based features. We chose these five feature representations as they are commonly used and showed good performance in the existing literature (Haque et al., 2022), (Xia et al., 2014) of cyber security and SE domain. NLP/text-based features that we chose are the count of word, character, noun, verb, adjective, adverb, pronoun and word density. Other NLP features such as punctuation (removed in pre-processing), and case (lower-cased in pre-processing) were not applicable as they were irrelevant to our corpus.

Table 4: Hyper-parameters of ML models and baselines (for best performing features of ML models).

Model	Feature	Tuned Hyper-parameter
NB	Word+Char	alpha: 0.06271, fit_prior: False
LR	Word+Char	C: 2.60, tolerance: 5.82e-05, fit_intercept: True, max_iter: 250, solver: lbfgs, warm_start: True
SVM	Char	C: 8.68, linear kernel, gamma: 3.56
RF	Word+Char	criterion: gini, max_depth: none, max_features: auto, n_estimators: 100
XGB	Word+Char	gamma: 0.16, learning_rate: 0.13, max_depth: 10, min_child_weight: 1.0, n_estimators: 27
SimScore	Word2Vec	window size: 5, min_count: 1, embedding dim: 300, workers: 3, skip-gram(sg): 1, hierarchical_softmax(hs): 1
APIRO	FastText	For fastText:- word_ngrams: 1, window: 5, embedding dim: 300, workers: 3, min_count: 1, skip-gram(sg): 1, hierarchical_softmax(hs): 1. For CNN:- embedding dim: 300, batch: 64, dropout: 0.5, epoch num: early stop with patience: 50, filter size:(3, 4, & 5), num of hidden node: 100, L2R: 0.0001, filter num: 100

For char level features, we chose an n-gram range of 2-4 characters as the vocabulary size did not increase after the value 4. Similarly, for word n-gram features, we chose the n-gram range of 2-4. We tuned each model's hyper-parameter based on Bayesian Optimisation (Feurer and Hutter, 2019) using the Hyperopt library (Bergstra et al., 2013). To find optimal values for parameters, we used Bayesian Optimisation for its robustness against the evaluation of noisy objective function and because it outperforms the other hyperparameter optimization approaches (e.g., random search) (Snoek et al., 2012). Each model for each feature representation was tuned individually. For conciseness, in Table 4 we report the tuned hyperparameter values of each model for only the best performing feature representation (detailed in Section 3.1). We used Scikit-learn (Scikit-learn, 2022a) and Gensim (Řehůřek and Sojka, 2010) to implement our ML models.

RQ2. Comparison with Baselines: To answer RQ2, we compared the performance of the best performing traditional ML model that we identified in RQ1 with two baselines. Firstly, to implement the state-of-the-art security tool API recommender called APIRO (Sworna et al., 2022), we used their fastText embed-

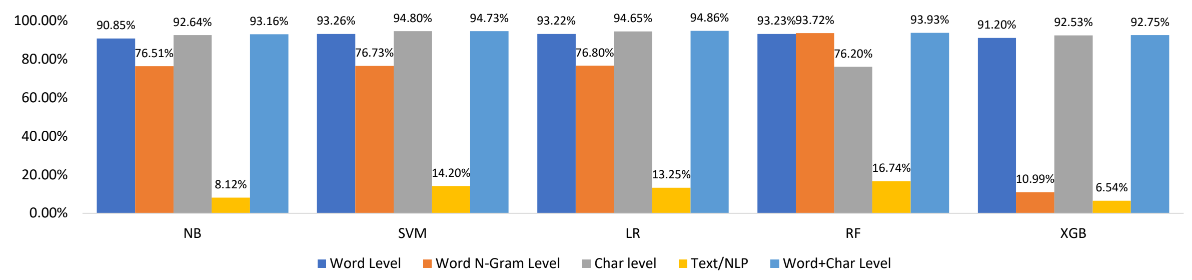

Figure 1: Acc of ML models with different feature representations.

ding (Bojanowski et al., 2017) based CNN approach. We first built a security tool API-specific fastText embedding on API_c corpus using Gensim (Řehůřek and Sojka, 2010). We built the CNN model based on the fastText embedding using Keras (Chollet et al., 2015) library. We followed the same hyper-parameter tuning process of APIRO (Sworna et al., 2022) and report our best parameter setting in Table 4.

To implement another baseline, SimScore, we used Word2Vec (Mikolov et al., 2013) and IDF-weighted cosine similarity score approach that is commonly adopted in the existing studies for Java API recommendation (Cai et al., 2019), (Huang et al., 2018), (Ye et al., 2016). Unlike the existing studies (Cai et al., 2019), (Huang et al., 2018) we do not consider the Stack Overflow (SO) data (i.e., we rely on API documentation), and our approach is not confined to a language-specific model as we consider diverse security tool's data for a unified solution. The parameter values for SimScore are shown in Table 4.

RQ3. Efficiency Evaluation: To answer RQ3, we evaluated the best performing ML model (identified in RQ1) with baselines of RQ2 in terms of required time (i.e., train, test) and resource utilization (e.g., model size, core hours in terms of CPU-time elapsed).

2.4 Evaluation Metrics

To answer RQ1 and RQ2, we evaluated our ML models and baselines using Accuracy (Acc), Mean Reciprocal Rank (MRR) and Mean Precision@K (MP@K). These metrics are widely used for API recommendation by the relevant literature (Huang et al., 2018), (Rahman et al., 2016), (Ye et al., 2016). To answer RQ2, we used delta (Fakhoury et al., 2018), which denotes the performance difference between two models. To answer RQ3, we evaluated the required time (i.e., train, test) and resource utilization (i.e., model size, maximum disk write, maximum disk read, core hours for CPU-time elapsed, User compute CPU, System I/O CPU and total CPU used) (Fakhoury et al., 2018).

Acc refers to the percentage of queries for which a recommender can recommend the actual API.

$$Acc(Q) = \frac{\sum_{q\in Q} Actual\_API(q)}{|Q|}\% \tag{1}$$

Here, Q refers to the list of all test queries. $Actual\_API(q)$ provides 1 if the actual API is recommended and 0 otherwise.

MRR denotes if the system returns the correct result at a high-ranked position. Reciprocal rank denotes the multiplicative inverse of the rank of the actual API in the top-K results returned by a recommender. MRR is the average for all search queries in the test data. Here, $Rank_q$ is the actual API's rank for an input query.

$$MRR(Q) = \frac{1}{|Q|}\sum_{q\in Q}\frac{1}{Rank_q} \tag{2}$$

MP@K denotes the mean of precision@K for all the test queries. A higher value of MP@K is expected for a low value of K.

$$MP@K(Q) = \frac{1}{|Q|}\frac{TPos@K}{(TPos@K)+(FPos@K)} \tag{3}$$

Here, $TPos@K$ denotes True_Positives@K and $FPos@K$ denotes False_Positives@K.

3 RESULTS

The RQ evaluation results are presented in this section.

3.1 RQ1. Traditional ML Models' Performance Investigation

Figure 1 shows the performance comparison of 5 ML models with 5 feature representations in terms of Acc for recommending security tools' API. Table 5 presents the performance comparison of the ML models with diverse features in terms of MRR, MP@1, MP@2 and MP@3. Our results show that LR using the combination of word and char level TF-IDF features outperforms all the other models by achieving 94.86% Acc, which indicates its ability to recommend correct APIs. Except for SVM, for all the other models, the

Table 5: Comparison of ML models using different features in terms of MRR, MP@1, MP@2 and MP@3.

Model	Feature	MRR	MP@1	MP@2	MP@3
NB	Word	94.15	90.85	47.95	32.50
	Word N-Gram	80.16	75.90	41.26	28.13
	Char	95.17	92.64	48.27	32.54
	Text/NLP	13.12	8.16	6.64	5.62
	Word+Char	95.61	93.16	48.47	32.68
RF	Word	96.01	93.17	48.82	32.91
	Word N-Gram	79.99	75.58	41.23	28.18
	Char	96.29	93.61	48.67	32.84
	Text/NLP	22.92	16.99	11.97	9.73
	Word+Char	96.29	94.03	48.76	32.87
SVM	Word	95.19	91.79	48.62	32.89
	Word N-Gram	78.35	73.47	40.36	27.78
	Char	96.77	94.58	49.02	33.01
	Text/NLP	20.03	13.09	10.18	8.42
	Word+Char	96.62	94.22	49.03	**33.05**
XGB	Word	94.35	91.20	47.99	32.48
	Word N-Gram	5.30	4.96	2.68	1.87
	Char	95.01	92.54	48.21	32.52
	Text/NLP	10.89	6.58	5.25	4.65
	Word+Char	95.20	92.76	48.26	32.57
LR	Word	95.90	93.23	48.90	32.96
	Word N-Gram	95.90	76.19	41.34	28.23
	Char	96.79	94.65	49.03	32.98
	Text/NLP	19.87	13.30	10.04	8.20
	Word+Char	**96.96**	**94.86**	**49.12**	**33.05**

best performing feature representation was the combination of word and char level feature, while the char level feature showed the best performance for SVM. This indicates that the combination of word and char level TF-IDF is the most beneficial feature representation for API description as this combination use both word and char for learning the semantics of the API description to provide better results. NLP/text-based features had the least Acc, MRR and MP@K.

The top 3 best performing ML models for security tool APIs recommendation are LR, SVM and RF. LR's high performance is an indication of a high correlation between the feature and target variable (Bailly et al., 2022). LR and SVM perform well for text data due to linear separation and the ability to generalize high dimensional features typical for text data (Fucci et al., 2019). Furthermore, SVM is capable of learning independent of the feature space and regularization allows it to resist over fitting. Whilst RF and XGB are both decision-tree-based ensemble models, RF outperformed XGB. One of the reasons can be the high number of APIs in our dataset. As for XGB, the number of trees to be trained is [number of iterations]*[number of APIs], whereas RF only requires [number of iterations] trees. RF classifier accumulates decisions from each tree for the final decision. Hence, RF has strong generalization achieving high Acc without over-fitting compared to XGB (Misra and Li, 2020). Overall, our results show that the LR model outperformed all the other ML models including the ensemble ones (i.e., RF and XGB). LR using the combination of word and char level features outperforms all the other models and achieves 96.96 MRR, which indicates its ability to recommend the correct API at a higher ranked position in the recommended list of APIs.

From these findings, we do not recommend the use of NLP/text-based features for recommending security tool APIs. We chose LR with the combination of word and char level features as the best performing ML model for recommending security tool APIs, as it outperformed all the other ML models with various feature representations in terms of all the evaluated effectiveness metrics.

Table 6: Comparison of best performing ML model with the baselines in terms of Acc, MP@K and MRR.

Model	Acc	MRR	MP@1	MP@2	MP@3
Best performing ML	**94.86**	**96.96**	**94.86**	**49.12**	**33.05**
SimScore	80.01	83.98	80.01	42.91	29.28
APIRO	94.50	96.70	94.50	49.04	32.90

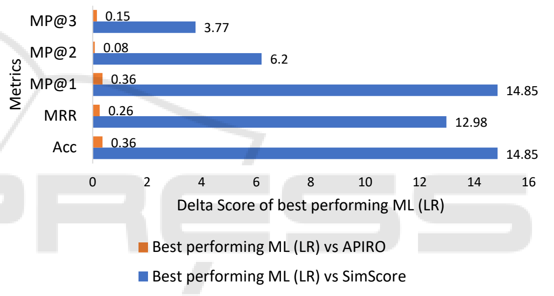

Figure 2: Delta Score of our best performing ML model with APIRO and SimScore baselines, where positive values indicate better performance of our ML model.

3.2 RQ2. Comparison with Baselines

Table 6 presents the Acc, MP@K and MRR for our best performing ML model (LR with the combination of word and char level features as discussed in RQ1) and the two baselines. Figure 2 shows the delta score which denotes the performance difference of our best performing ML model with SimScore and APIRO, respectively for each metric. In Figure 2, any horizontal bar above zero represents that our best performing ML model performs better and the higher the value of delta, the better our best performing ML model performs. Any bar below zero indicates that the baseline performs better. As shown in Figure 2, all the bars are above zero representing that our best performing ML model outperforms baselines in terms of Acc, MRR, MP@1, MP@2 and MP@3.

Our experimental result shows that our best performing ML model achieved deltas in the range of 0<delta<0.36. A prior study (Fakhoury et al., 2018)

Table 7: Comparison of our best performing ML model with the baselines in terms of resource utilization.

Approach	CPU time elapsed (core hours)	Total CPU used (core hours)	User CPU (Compute): (core hours)	System CPU (I/O): (core hours)	Model size (GB)	Max Disk Write (GB)	Max Disk Read (GB)
Best performing ML (LR)	6.27	0.74	0.63	0.11	0.12	0.11	0.03
SimScore	166.17	16.80	15.90	0.91	0.01	0.62	38.47
APIRO	24.14	18.11	17.89	0.22	5.10	12.63	0.42
Improve. SimScore	96.23%	95.60%	96.04%	87.91%	-1100%	82.26%	99.92%
Improve. APIRO	74.03%	95.91%	96.48%	50.00%	97.65%	99.13%	92.86%

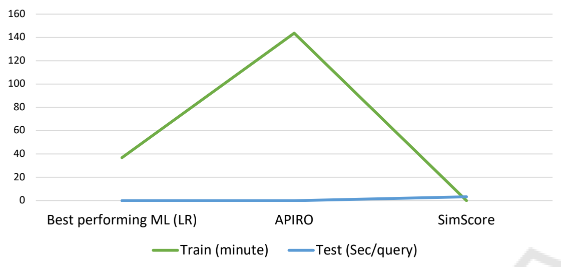

Figure 3: Time comparison of our best performing ML model and baselines for security tool API recommendation.

recommended the applicability of ML over DL in SE by achieving deltas for their ML model in the range of -0.1<delta<0.3 for linguistic smell detection. In line with the resultant deltas (Fakhoury et al., 2018), our achieved resultant deltas make the applicability suitable for using ML over DL (i.e., APIRO) for security tool API recommendation. Besides, our best performing ML model significantly outperformed SimScore baseline by achieving 18.57% and 15.45% improvement in terms of Acc and MRR. It indicates that our best performing ML model can recommend correct APIs at a higher ranked position in the recommended list of APIs.

Our result shows that our best performing ML model outperforms the SimScore and DL-based APIRO baselines. Hence, our best performing ML model is recommended for security tool API recommendation.

3.3 RQ3. Efficiency Evaluation

Table 7 shows the comparison of the resource utilization of our best performing ML model (LR with word and char level features) with baselines (i.e., APIRO and SimScore). The memory (in terms of GB) consumption reduction percentages by our best performing ML model compared to APIRO are 97.65% model size, 99.13% maximum disk write and 92.86% maximum disk read. Our best performing ML model also significantly reduces the required core hours compared to APIRO in terms of 74.03% CPU time elapsed, 96.48% User compute CPU, 50.00% System I/O CPU and 95.91% total CPU used. Hence, DL-based APIRO is highly resource consuming as it consumes more resources than our best performing ML model. Similarly, another baseline SimScore also requires more memory than our best performing ML model in terms of 82.26% maximum disk write and 99.92% maximum disk read, except for model size, where SimScore's model size is smaller than our best performing ML model. However, the Acc of SimScore is 18.57% less than our best performing ML model and SimScore requires more core hours than our best performing ML model with a percentage increase of 96.23% CPU time elapsed, 96.04% User compute CPU, 87.91% System I/O CPU and 95.60% total CPU used. Hence, the drastically high resource utilization of the DL-based approach strongly justifies the use of our less resource consuming ML approach.

The time for training and testing each model is shown in Figure 3. For training, DL-based APIRO requires 291.50% more time than our best performing ML model. Though SimScore requires less training time than our best performing ML model, the Acc of SimScore is 18.57% lower than our best performing ML model as mentioned earlier. Besides, SimScore requires 3.40 seconds per query, whereas our best performing ML model and APIRO both require less than a second to answer a query in terms of testing time. SimScore requires the maximum testing time compared to the other models as it calculates the cosine similarity score for the query with each API description of the training corpus.

Our results show that DL-based APIRO requires a significantly large amount of resources (e.g., memory, CPU time, Disk read and write) and significantly high training time compared to our best performing ML model. Hence, considering both the effectiveness (e.g., Acc, MRR) and the efficiency (e.g., time and resources), our best performing ML model performs significantly better for security tool API recommendation than baselines and provides significant cost-cutting opportunities to be adopted in the real world SOCs.

4 THREATS TO VALIDITY

The representativeness of the experimental data to the real world SOC environment is the primary threat to validity. We address this threat by conducting experiments on data collected from 7 real world security tools that are widely used in SOCs.

Enriching the security tool API corpus by adding more security tools' data is a continuous process. Our dataset may not cover each possible query. Besides, answering queries having extreme rare terms can be difficult for our approach. However, our generalized approach enables re-training of the model for dealing with such queries, where additional augmented text and new augmentation techniques can be used to handle the extreme rare terms.

To ensure a fair and valid comparison, we implemented our ML models and baselines in the same environment using the same pre-processing steps on the same dataset. We ran each experiment ten times and reported the average to mitigate the experimental randomness of results, which is a common repeat scheme for learning-based models (Wang et al., 2019). The chosen optimal hyper-parameters for traditional ML models may not ensure the best result to recommend suitable APIs. To minimize this threat, we used 10-fold cross-validation and the state-of-the-art Bayesian Optimisation for choosing optimal hyper-parameters.

5 IMPLICATIONS

Our results generated implications for both practitioners and researchers based on our analysis.

5.1 Implication for Practitioners

Resource constraint is considered as one of the significant challenges for utilizing AI-based (e.g., ML, DL, NLP) solutions (Fu and Menzies, 2017). We present important and deep insights into the comparative analysis of ML, DL and similarity score-based approaches for security tool API recommendation. The findings can enable practitioners to understand the resource utilization of the investigated approaches before choosing a particular AI-based approach to automating the security tool APIs recommendation. Instead of preferring DL-based solution, practitioners may also consider the use of traditional ML models, specifically Small to Medium-sized Enterprises (SMEs) with limited resource and time allocation for ensuring security (CSCRC, 2022). Since the use of DL is expensive, most SOC will desire to use a simple ML approach. As LR achieved better performance than DL with relatively less amount of time and resources, practitioners may consider LR for automated security tool API recommendation. Besides, the adoption of new security tools and APIs requires re-training of a model. As ML showed significantly lower training time compared to the DL approach, ML is a viable option for practitioners to re-train a model to cope with the evolving threat landscape with significant cost-cutting opportunities that helps achieve sustainable design.

5.2 Implication for Researchers

Our best performing ML model motivates sustainable design presenting a cautionary tale to researchers to keep resource constraints in mind for real world applicability, as cost, resource and time constraints may present significant barriers to the industrial adoption of a proposed solution. This cautionary recommendation based on a completely different task (i.e., security tool API recommendation) strengthens this suggestion by the prior studies (Fakhoury et al., 2018), (Fu and Menzies, 2017). In the future, researchers can focus on automated execution of the recommended APIs for executing an Incident Response Plan (IRP) in response to a security incident. Our best performing ML model can also be explored for query answer recommendations from other artifacts of security tools (e.g., user guide and tutorial).

6 RELATED WORK

This section summarizes the related work on security incident response and API recommendation.

6.1 Cyber Security Incident Response

Different SOAR platforms (e.g., Swimlane (Swimlane, 2022), D3 (D3, 2017) and ThreatConnect (ThreatConnect, 2019)) leverage APIs for incident response. However, developers and users of these SOAR platforms manually look for the required APIs from the documentation to create, update and execute IRPs (Sworna et al., 2022). APIRO is a DL-based foundation framework for security tool API recommendation to support incident response using a CNN approach (Sworna et al., 2022). In contrast to our study, APIRO does not focus on resource utilization to mitigate SOC's cost and resource constraints (CSCRC, 2022).

6.2 API Recommendation in SE

The existing studies on programming language specific APIs recommendation are mostly code, Stack Overflow (SO) or documentation based approaches. For code based studies, a set of studies (He et al., 2021), (Ling et al., 2020), (Ling et al., 2019), (Xu et al., 2018), (Gu et al., 2016), (Raghothaman et al., 2016), (Chan et al., 2012), (McMillan et al., 2011) used open source GitHub code repositories for recommending API usages and code snippets. These code based approaches perform time consuming complex code analysis.

The SO based approach is used for API recommendation (Wu et al., 2021), (Zhou et al., 2021), (Cai et al., 2019), (Huang et al., 2018), (Rahman et al., 2016). However, SO is slow at covering new APIs (Parnin et al., 2012) and SO data is less reliable as it may cause security flaws in code due to considering unreliable suggestions (Acar et al., 2016). To mitigate these issues, we use the comprehensive security tool APIs documentation.

The code based and SO based approaches can only cover the frequently used and discussed APIs (Xie et al., 2020). Hence, our approach uses API documentation so that our approach is not unable to recommend APIs that are less frequently used. Besides, a study (Xie et al., 2020) consider API documentation to manually identify, categorize verbs and manually generate verb-phrase patterns for matching functionality verb-phrases of API description and query. It is highly time and labor-intensive to perform these huge manual analysis on each API documentation of the wide range of security tools that a SOC can utilize. Thus, we avoid these manual efforts and adopt automated data augmentation and the ML-based approach to recommend security tool API. Another study (Fucci et al., 2019) used the ML-based approach to annotate knowledge types (e.g., purpose, quality and environment) to API documentation, in contrast, we recommend security tool API from diverse API documentation.

7 CONCLUSION

SOCs face cost, time and resource constraints to respond to numerous security incidents. Hence, our study inspires Green-AI and sustainable design by presenting a cautionary tale to keep resource constraints in mind for real world applicability of security tools' API recommendation. We performed an empirical study comparing 5 simple ML models with various feature representations compared to the complex DL models aiming at both efficiency and effectiveness for recommending security tools' API. Using APIs of seven widely used real world security tools, our best performing ML model achieved an exemplary reduction of resource and time with better Acc and MRR compared to baselines including the state-of-the-art DL-based approach. Our empirically derived best performing ML model with its tremendous time and resource efficiency including better effectiveness makes it a viable approach to be adopted in real world SOC with significant cost-cutting opportunities. Our future research will focus on extending our best performing ML model by using the recommended APIs for automated execution of the security incident response plans in SOC.

ACKNOWLEDGEMENTS

The work has been supported by the Cyber Security Research Centre Limited whose activities are partially funded by the Australian Government's Cooperative Research Centres Programme. This work has also been supported with super-computing resources provided by the Phoenix HPC service at the University of Adelaide.

REFERENCES

Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M. L., and Stransky, C. (2016). You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305. IEEE.

Bailly, A., Blanc, C., Francis, É., Guillotin, T., Jamal, F., Wakim, B., and Roy, P. (2022). Effects of dataset size and interactions on the prediction performance of logistic regression and deep learning models. *Computer Methods and Programs in Biomedicine*, 213:106504.

Beautifulsoup (2020). Beautiful soup package. https://tinyurl.com/yf27edhf. Accessed October 1, 2022.

Bergstra, J., Yamins, D., and Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR.

Biswas, S., Wardat, M., and Rajan, H. (2021). The art and practice of data science pipelines: A comprehensive study of data science pipelines in theory, in-the-small, and in-the-large. *arXiv preprint arXiv:2112.01590*.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Cai, L., Wang, H., Huang, Q., Xia, X., Xing, Z., and Lo, D. (2019). Biker: a tool for bi-information source based api method recommendation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1075–1079.

Calero, C. and Piattini, M. (2015). Introduction to green in software engineering. In *Green in Software Engineering*, pages 3–27. Springer.

Chan, W.-K., Cheng, H., and Lo, D. (2012). Searching connected api subgraph via text phrases. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11.

Chatterjee, P., Damevski, K., and Pollock, L. (2021). Automatic extraction of opinion-based q&a from online developer chats. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1260–1272. IEEE.

Choetkiertikul, M., Dam, H. K., Tran, T., Pham, T., Ghose, A., and Menzies, T. (2018). A deep learning model for estimating story points. *IEEE Transactions on Software Engineering*, 45(7):637–656.

Chollet, F. et al. (2015). Keras. https://github.com/fchollet/keras. Accessed October 20, 2022.

CSCRC, CSIRO's Data61, C. (2022). Small but stronger: Lifting sme cyber security. https://tinyurl.com/2juxjnk6. Accessed October 21, 2022.

Cuckoo (2019). Cuckoo: Automated malware analysis. https://cuckoosandbox.org/. Accessed October 20, 2022.

D3 (2017). Enterprise incident & case management solution for security orchestration, automation & response. https://tinyurl.com/5bcwkuk4. Accessed October 20, 2022.

Fakhoury, S., Arnaoudova, V., Noiseux, C., Khomh, F., and Antoniol, G. (2018). Keep it simple: Is deep learning good for linguistic smell detection? In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 602–611.

Feurer, M. and Hutter, F. (2019). Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, Cham.

Fu, W. and Menzies, T. (2017). Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, pages 49–60.

Fucci, D., Mollaalizadehbahnemiri, A., and Maalej, W. (2019). On using machine learning to identify knowledge in api reference documentation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 109–119.

Gu, X., Zhang, H., Zhang, D., and Kim, S. (2016). Deep api learning. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 631–642.

Haque, M. U., Kholoosi, M. M., and Babar, M. A. (2022). Kgsecconfig: A knowledge graph based approach for secured container orchestrator configuration. In *2022 IEEE 29th International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE.

He, X., Xu, L., Zhang, X., Hao, R., Feng, Y., and Xu, B. (2021). Pyart: Python api recommendation in real-time. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1634–1645. IEEE.

Huang, Q., Xia, X., Xing, Z., Lo, D., and Wang, X. (2018). Api method recommendation without worrying about the task-api knowledge gap. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 293–304. IEEE.

Instinct, D. (2021). Research study by deep instinct. https://www.helpnetsecurity.com/2021/02/17/malware-2020/. Accessed February 10, 2022.

Le, T. H. M., Hin, D., Croft, R., and Babar, M. A. (2020). Puminer: Mining security posts from developer question and answer websites with pu learning. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 350–361.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Limacharlie (2022). Official limacharlie page. https://www.limacharlie.io/. Accessed October 20, 2022.

Ling, C., Lin, Z., Zou, Y., and Xie, B. (2020). Adaptive deep code search. In *Proceedings of the 28th International Conference on Program Comprehension*, pages 48–59.

Ling, C.-Y., Zou, Y.-Z., Lin, Z.-Q., and Xie, B. (2019). Graph embedding based api graph search and recommendation. *Journal of Computer Science and Technology*, 34(5):993–1006.

Ma, E. (2019). Nlp augmentation. https://github.com/makcedward/nlpaug. Accessed October 20, 2022.

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

McMillan, C., Grechanik, M., Poshyvanyk, D., Xie, Q., and Fu, C. (2011). Portfolio: finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 111–120.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

MISP (2022). Malware information sharing platform. https://www.misp-project.org. Accessed October 20, 2022.

Misra, S. and Li, H. (2020). Chapter 9 - noninvasive fracture characterization based on the classification of sonic wave travel times. In Misra, S., Li, H., and He, J., editors, *Machine Learning for Subsurface Characterization*, pages 243–287. Gulf Professional Publishing.

Mou, L., Li, G., Zhang, L., Wang, T., and Jin, Z. (2016). Convolutional neural networks over tree structures for programming language processing. In *Thirtieth AAAI conference on artificial intelligence*.

Muncaster, P. (2021). Organizations now have 76 security tools to manage. https://www.infosecurity-magazine.com/news/organizations-76-security-tools/. Accessed October 20, 2022.

Nguyen, T. D., Nguyen, A. T., Phan, H. D., and Nguyen, T. N. (2017). Exploring api embedding for api usages and applications. In *2017 IEEE/ACM 39th International*

Conference on Software Engineering (ICSE), pages 438–449. IEEE.

PAN (2019). Palo alto networks: The state of soar report. https://start.paloaltonetworks.com/the-2019-state-of-soar-report.html. Accessed October 21, 2020.

PAN (2022). Palo alto networks: Cortex xsoar. https://xsoar.pan.dev. Accessed September 3, 2022.

Parnin, C., Treude, C., Grammel, L., and Storey, M.-A. (2012). Crowd documentation: Exploring the coverage and the dynamics of api discussions on stack overflow. *Georgia Institute of Technology, Tech. Rep*, 11.

Phantom (2021). Splunk phantom: Harness the full power of your security investments with security orchestration, automation and response. https://tinyurl.com/4493k5nk. Accessed September 3, 2022.

Raghothaman, M., Wei, Y., and Hamadi, Y. (2016). Swim: Synthesizing what i mean-code search and idiomatic snippet synthesis. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 357–367. IEEE.

Rahman, M. M., Roy, C. K., and Lo, D. (2016). Rack: Automatic api recommendation using crowdsourced knowledge. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 349–359. IEEE.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA.

Robillard, M. P. and DeLine, R. (2011). A field study of api learning obstacles. *Empirical Software Engineering*, 16(6):703–732.

Scikit-learn (2022a). Api reference. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics. Accessed October 20, 2022.

Scikit-learn (2022b). Cross-validation: evaluating estimator performance. https://tinyurl.com/4ue9k44b. Accessed October 20, 2022.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.

Snort (2020). Snort users manual 2.9.16. https://tinyurl.com/2p8ferjm. Accessed October 20, 2022.

Splunk (2022). Splunk: Siem, aiops, application management. https://www.splunk.com/. Accessed October 20, 2022.

Steven Bird, E. K. and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media Inc.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Swimlane (2022). Security orchestration, automation and response (soar) capabilities. https://tinyurl.com/2p82wjvu. Accessed October 20, 2022.

Sworna, Z. T., Islam, C., and Babar, M. A. (2022). Apiro: A framework for automated security tools api recommendation. *ACM Trans. Softw. Eng. Methodol.* Just Accepted.

ThreatConnect (2019). Everything you need to know about soar. https://tinyurl.com/yfhp7sfx. Accessed October 20, 2022.

Vielberth, M., Böhm, F., Fichtinger, I., and Pernul, G. (2020). Security operations center: A systematic study and open challenges. *IEEE Access*, 8:227756–227779.

Wang, S., Phan, N., Wang, Y., and Zhao, Y. (2019). Extracting api tips from developer question and answer websites. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 321–332. IEEE.

Wazuh (2022). Wazuh website. https://wazuh.com. Accessed October 20, 2022.

White, M., Vendome, C., Linares-Vásquez, M., and Poshyvanyk, D. (2015). Toward deep learning software repositories. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 334–345.

Wu, D., Jing, X.-Y., Zhang, H., Zhou, Y., and Xu, B. (2021). Leveraging stack overflow to detect relevant tutorial fragments of apis. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 119–130.

Xia, X., Lo, D., Qiu, W., Wang, X., and Zhou, B. (2014). Automated configuration bug report prediction using text mining. In *2014 IEEE 38th Annual Computer Software and Applications Conference*, pages 107–116.

Xie, W., Peng, X., Liu, M., Treude, C., Xing, Z., Zhang, X., and Zhao, W. (2020). Api method recommendation via explicit matching of functionality verb phrases. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1015–1026.

Xu, C., Sun, X., Li, B., Lu, X., and Guo, H. (2018). Mulapi: Improving api method recommendation with api usage location. *Journal of Systems and Software*, 142:195–205.

Ye, X., Shen, H., Ma, X., Bunescu, R., and Liu, C. (2016). From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th international conference on software engineering*, pages 404–415.

Yenigalla, P., Kar, S., Singh, C., Nagar, A., and Mathur, G. (2018). Addressing unseen word problem in text classification. In *International Conference on Applications of Natural Language to Information Systems*, pages 339–351. Springer.

Zeek (2020). Zeek: An open source network security monitoring tool. https://zeek.org/. Accessed October 20, 2022.

Zhou, Y., Yang, X., Chen, T., Huang, Z., Ma, X., and Gall, H. C. (2021). Boosting api recommendation with implicit feedback. *IEEE Transactions on Software Engineering*.