Journal of Innovation & Knowledge 8 (2023) 100401

Journal of Innovation & Knowledge

https://www.journals.elsevier.com/journal-of-innovation-and-knowledge

Facilitating innovation in the API economy: Privacy-enhanced and novelty-aware API recommendation for enterprises

Baogui Xin[a], Chao Yan[a,b], Yuxuan Cao[b], Muhammad Bilal[c,*]

[a] College of Economics and Management, Shandong University of Science and Technology, Qingdao, 266590, China
[b] School of Computer Science, Qufu Normal University, Rizhao, 276826, China
[c] Department of Computer Engineering, Hankuk University of Foreign Studies, Yongin-si, 17035, South Korea

ARTICLE INFO

Article History:
Received 13 June 2022
Accepted 14 June 2023
Available online 20 June 2023

Keywords:
API recommendation
Popularity bias
Privacy preservation
Recommendation novelty

ABSTRACT

Web APIs provide enterprises with a new way of driving innovations of new technology with limited resources. API recommendations greatly alleviate the selection burdens of enterprises in identifying potential useful APIs to meet their business demands. However, these approaches disregard the privacy leakage risk in cross-platform collaboration and the popularity bias in recommendation. To address these issues, first, we introduce MinHash, an instance of locality-sensitive hashing, into a collaborative filtering technique and propose a novel, privacy-enhanced, API recommendation approach. Second, we present a simulation algorithm to analyze the popularity bias in API recommendation. Third, we mitigate popularity bias by improving the novelty of recommendation results with an adaptive reweighting mechanism. Last, comprehensive experiments are conducted on a real-world dataset collected from ProgrammableWeb. Experimental results show that our proposed approach can effectively preserve usage data privacy and mitigate popularity bias at a minimum cost in accuracy.

Introduction

The emergence of web application programming interfaces (referred to as APIs hereafter) defines a new way of making applications easier to develop, driving innovations of new technology with limited resources (Calero et al., 2019; Hu et al., 2021; Wulf & Blohm, 2020). Because of their advantages in speed, ease, and portability in data exchange, many enterprises offer their services utilizing APIs. An increasing number of enterprises outsource specific business requirements to APIs (Catlett et al., 2020; Jensen & Ashby, 2018; Sánchez et al., 2019). This trend is referred to as the API economy (Bonardi et al., 2016). With the growing prosperity of the API economy, the number of APIs has dramatically grown (Evans & Basole, 2016). According to ProgrammableWeb[1] (referred to as PW hereafter), the most prominent API repository, there were more than 24,500 APIs on the website as of May 2022, which is a thirtyfold increase since 2008 (refer to Fig. 1). All the APIs are derived from 507 categories across a wide range of market sectors, e.g., financial, data, social, tools, e-commerce, and payments (see Fig. 2).

The vast number of APIs provide enterprises with a wide range of choices. However, enterprises also face heavy burdens in selecting suitable APIs to meet their demands (Vijayakumar et al., 2022). Fortunately, recommendation techniques (e.g., collaborative filtering) have demonstrated effectiveness in alleviating such burdens (Zhou et al., 2020). A series of API recommendation approaches have been proposed for different scenarios (Bai et al., 2020; Kang et al., 2020; Qi et al., 2022; Yan et al., 2021). However, two significant challenges arise for current API recommendation approaches.

First, many recommendation approaches (e.g., matrix factorization) assume that the historical usage data employed for API recommendation are centrally stored. However, in practice, these data are often owned by different platforms. Moreover, these platforms are reluctant to share data with others because of privacy concerns (Liang et al., 2020; Yang et al., 2022). Consequently, the risk of privacy leakage heavily reduces the possibility of cross-platform collaboration, which decreases the accuracy of recommendation.

Second, because of the intrinsic features of recommendation approaches and training sets, most recommendation methods suffer from popularity bias issues, i.e., popular APIs are overly recommended to enterprises. However, recommending popular APIs to enterprises is trivial. Enterprises may have been aware of popular APIs and decided whether they met their business demands. In

* Corresponding author.
E-mail address: m.bilal@ieee.org (M. Bilal).
[1] https://www.programmableweb.com

https://doi.org/10.1016/j.jik.2023.100401

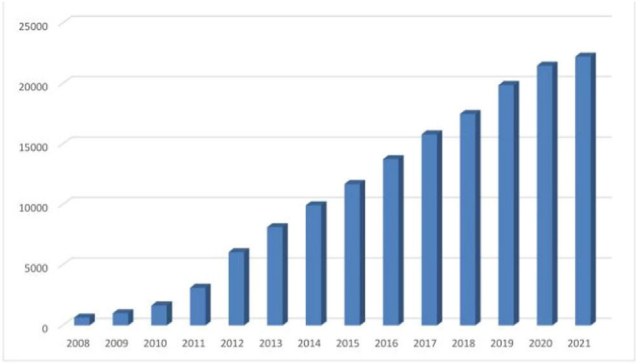


Fig. 1. Evolution of the number of submitted APIs in PW.

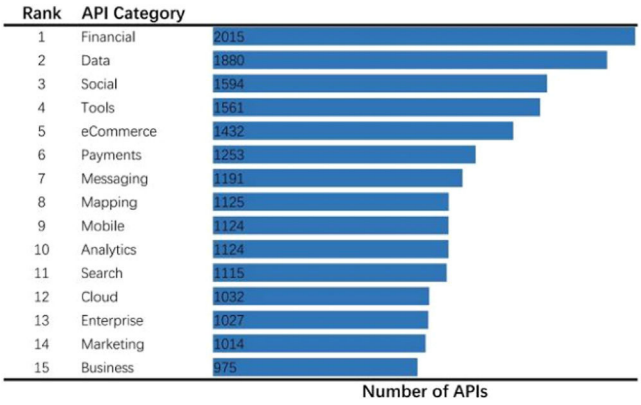


Fig. 2. Open APIs by top 15 sectors in PW.

addition, enterprises tend to adopt novel APIs that differentiate from their competitors to innovate at a faster rate than competitors.

Given the abovementioned challenges, we propose a novel API recommendation approach named *NAPIRec*, which introduces MinHash to the traditional collaborative filtering technique to preserve data privacy and promote cross-platform collaboration. Furthermore, we present an adaptive weighting mechanism to mitigate the popularity bias of our proposed approach. To the best of our knowledge, few studies have attempted to employ MinHash to preserve data privacy in API recommendations. Moreover, few studies have focused on the popularity bias issue in API recommendations. The significant contributions of this paper include the following:

- We analyze the usage frequency of APIs in PW to reveal the long-tail distribution in API usage. Then, we design a simulation algorithm and demonstrate the popularity bias produced by a classical collaborative filtering approach.
- We introduce MinHash into a collaborative filtering technique to preserve API usage privacy in API recommendation. We present an adaptive weighting mechanism to mitigate popularity bias and improve the novelty of recommendation results.
- We conduct a series of experiments on a real-world dataset to validate the superiority of our proposed method compared with state-of-the-art approaches.

The remainder of this paper is organized as follows: first, we review related works in API recommendation. Second, we formulate the definition of API recommendation and analyze the popularity bias issue in API recommendation. *NAPIRec* is introduced and discussed in the "*Methodology*" section. The "*Results and Discussion*" section reports the experimental results conducted on a real-world dataset and discusses its threats to validity. Last, we conclude this paper and outline our future work.

Literature review

With the wide adoption of APIs in enterprises, API recommendation has attracted attentions from both researchers and practitioners. A series of API recommendation approaches have emerged, as shown in Table 1. Generally, these approaches are divided into explicit feedback-based approaches and implicit feedback-based approaches. For explicit feedback-based approaches, user preference is explicitly represented as keywords (Gong et al., 2021; Qi et al., 2020, 2022; Xiao et al., 2020) or textual descriptions (Gao & Wu, 2017; Yan et al., 2021).

However, to achieve a better user experience, platforms often need to make recommendations without knowing explicit user preferences. Therefore, implicit feedback from historical usage information, such as QoS (Qi et al., 2018; Wang et al., 2021) and historical invocation records, is employed to capture user preference. As the QoS information of APIs is generally incomplete and hard to obtain, most API recommendation approaches focus on utilizing historical invocation records to capture user preference (Yan et al., 2021; Yao et al., 2021).

Recommendation approaches based on implicit feedback usually adopt collaborative filtering methods (Wang et al., 2022), e.g., matrix factorization (He et al., 2022; Yao et al., 2021). These approaches assume that the recommendation base, e.g., historical usage data, is centralized on one platform. However, in practice, historical usage data are distributed across different platforms. Moreover, platforms tend not to share data because of privacy concerns. Qi et al. (2018) introduced locality-sensitive hashing (LSH) to preserve the data privacy of service providers in web service recommendations. However, Qi's recommendation algorithm is based on continuous QoS data, which is unsuitable for discrete API usage data. To the best of our knowledge, few researchers have focused on the problem of privacy preservation in API recommendation.

Table 1
Differential API recommendation approaches.

Study	Preference Representation.	Model	Privacy-Aware	Considering Popularity Bias
Xiao et al. (2020)	Keywords	Deep Neural Network	No	No
Qi et al. (2022, 2020)	Keywords	Minimal Steiner Tree	No	No
Cheng et al. (2020)	Keywords	Graph Search	No	No
Gong et al. (2021)	Keywords	Minimal Steiner Tree	No	No
Gao and Wu (2017)	Plain text	Clustering	No	No
Yan et al. (2021)	Plain text	GCN	No	No
Yao et al. (2021)	Implicit	Matrix Factorization	No	Yes
Wang et al. (2021)	Implicit	Cover Tree	No	No
He et al. (2022)	Implicit	Matrix Factorization	No	Yes
Qi et al. (2018)	Implicit	LSH + User-based CF	Yes	No
This study	Implicit	MinHash +User-based CF	Yes	Yes

Some researchers have noticed the popularity bias issue in recommendation and conducted a series of studies. He et al. (2022) investigated the popularity bias in matrix factorization-based, third-party library usage prediction, and neutralized the popularity bias by employing an adaptive mechanism. Gong et al. (2021) improved the diversity of API compositions by introducing sampling techniques to API recommendations. However, from the perspective of innovation, diversity is not equal to novelty, which is more likely to lead to innovation. Little attention has been paid to recommending novel APIs for enterprises. Compared with the previous research works, we propose a recommendation algorithm to meet the demands of enterprises for novel and valuable APIs while preserving their data privacy.

Popularity bias in API recommendation

API recommendation

Most enterprise applications invoke several APIs to implement their business requirements, e.g., online payment and map navigation. Given an enterprise application set $E = \{e_1, e_2, \ldots, e_m\}$, and an API set $A = \{a_1, a_2, \ldots, a_n\}$ which are registered in platforms, we represent the API usage data of all the enterprise applications in E as a $m \times n$ matrix (referred to as the usage matrix hereafter), as shown in Fig. 3. For an enterprise application $e_i \in E$, if API a_j is consumed by e_i, $u_{i,j}$ is set to 1. Otherwise, $u_{i,j}$ is set to 0. The aim of API recommendation is, for an enterprise application e_i, to select K APIs $\{a'_1, a'_2, \ldots, a'_K\} \subseteq A$ that are not consumed by e_i but are most likely applicable to e_i.

As typical representatives of light applications in the API economy, mashups are created by integrating APIs from different providers to implement specific business requirements. In our preliminary study, we collected 6530 mashups from PW. We discover that some API pairs cooperate in many mashups. Figure 4 shows the cumulative distribution of the cooperation frequency of API pairs, i.e., times invoked by the same mashup. For all the API pairs that collaborate in at least one mashup, approximately 35% of API pairs cooperate in more than five mashups, and more than 20% API pairs appear in more than ten mashups. The API usage patterns inspire a new method to recommend potential useful APIs for enterprises utilizing collaborative filtering.

Popularity bias

Collaborative filtering-based approaches usually suffer from the popularity bias problem, i.e., a small fraction of APIs consumed by most enterprises (named popular APIs) dominate the recommendation list. If this situation continues, APIs consumed by all the enterprise applications will concentrate on a tiny number of popular APIs, which may lead to two problems. First, as many enterprises have been aware of the most popular APIs, they will not benefit much from a recommendation list dominated by popular APIs. Second, newly published APIs have little chance of being seen and consumed by enterprises, hindering innovation in the API economy.

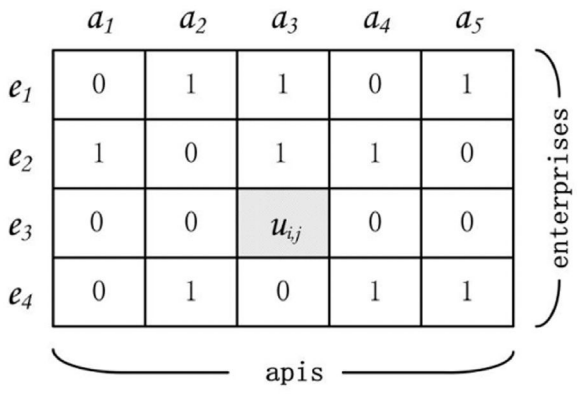


Fig. 3. An example of Enterprise-API matrix.

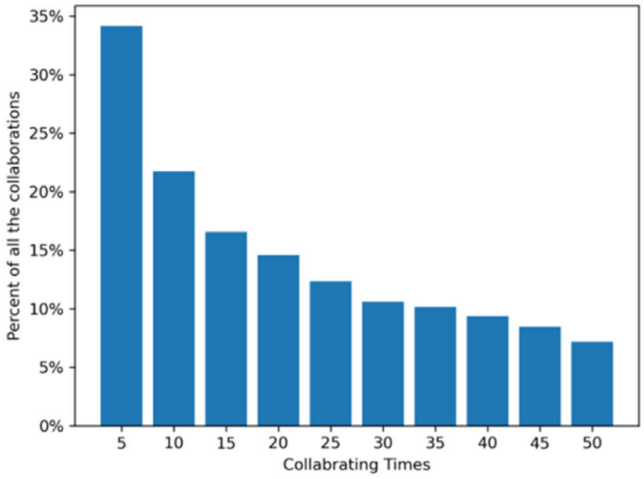


Fig. 4. Cumulative distribution of cooperating frequency of API pairs.

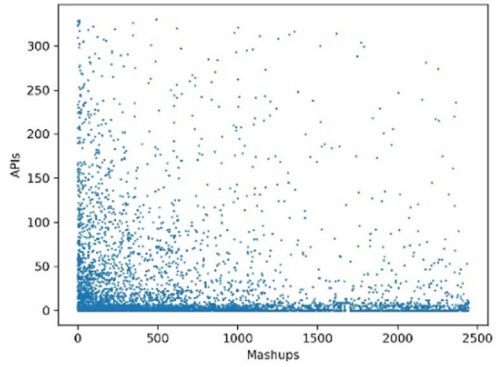
We conduct experiments on a real-world dataset from PW to investigate the popularity bias in API recommendation. To simulate the scenario of recommending APIs for enterprise applications employing historical API usage information, we choose mashups that invoke more than two APIs and APIs consumed by more than one mashup. Then, we build a 2445 × 604 usage matrix M. Detailed experimental settings are discussed in the "*Results and Discussion*" section. We propose an iterative simulation algorithm to investigate the longitudinal effect of collaborative filtering recommendation, as shown in Algorithm 1. We assume that enterprises make decisions entirely relying on the recommendation results. Enterprises consume the first API in the recommendation list during each simulation period. We update the usage matrix with new usage data at the end of each iteration. Then, we make recommendations employing the updated mashup-API matrix. We plot the API usage matrix with a scatter diagram, as shown in Fig. 5(a). Each row indicates a mashup, and each column indicates an API. Data is sorted by the usage frequency of APIs and the number of APIs consumed by mashups. Therefore, the first row in the scatter diagram represents the mashup that consumed the most APIs, and the first column represents the API invoked by the most mashups.

Figure 5(a) shows the initial distribution of API usage data. The initial distribution reflects a snapshot of realistic, long-tail distribution, where a small fraction of APIs is consumed by numerous mashups. The results are consistent with the distribution of the usage frequency of APIs, as shown in Fig. 6.

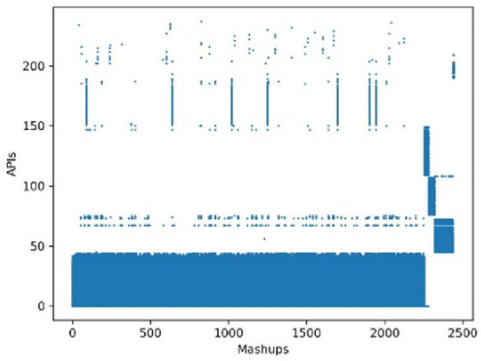
Figure 5(b) depicts the structural distribution of APIs newly consumed by mashups over 40 simulation periods. The distribution

Algorithm 1
Algorithm of Iterative Simulation Procedure.

Input : Mashup-API matrix: M, simulation periods: T
Output : Mashup-API matrix of newly consumed APIs
1: M'=M
2: **for** $t = 1$ *to* T **do**
3: **for** each row m in M' **do**
4: Predict the values of unused APIs in m
5: Select API j with the largest predicted value
6: $M'[i, j] = 1$
7: ***end for***
8: ***end for***
9: **return** M'-M



(a) The original mashup-API matrix



(b) The newly added APIs iterated over 20 times based on the recommender system

Fig. 5. Distribution scatter plots of recommended APIs.

indicates that the newly consumed APIs are condensed into a small set of APIs. The reason is that the recommendation results of the collaborative filtering approach mainly depend on the characteristics of historical data, e.g., density and value distribution (Adomavicius & Zhang, 2012). As API usage data follow the long-tail distribution (Park & Tuzhilin, 2008), the recommendation results are more likely to show high skewness. Moreover, increased reliance on the recommendation results leads to a skewed distribution of API usage data, producing more condensed recommendation results.

However, enterprises may not benefit much from popular APIs because they may already be familiar with them. In addition, from the perspective of innovation, recommendation results offering novelty and serendipity are more attractive to enterprises.

Methodology

MinHash

MinHash is also known as a minwise independent permutation locality-sensitive hashing scheme. Locality-sensitive hashing (LSH hereafter) (Gionis et al., 1999) is an approximate nearest neighbor search method based on hashing. Letting $D(\cdot,\cdot)$ denote a distance function of two elements from a set P, and $Pr(\cdot)$ represent the probability that an event holds.

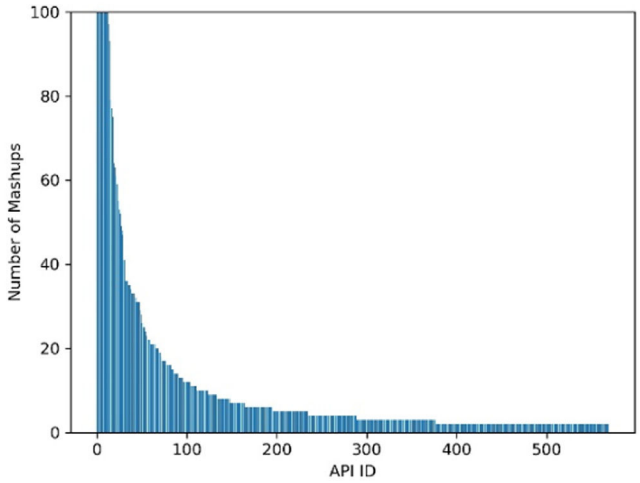


Fig. 6. Long-tail distribution of the usage of APIs.

Definition 1. A hash function $h(\cdot)$ is referred to as (d_1, d_2, p_1, p_2)-sensitive for $D(\cdot,\cdot)$ if for any $x, y \in P$:

- *if* $D(x,y) \leq d_1$, $Pr\Big(h(x) = h(y)\Big) \geq p_1$
- *if* $D(x,y) \geq d_2$, $Pr\Big(h(x) = h(y)\Big) \leq p_2$

Due to its advantage in compact storage and efficient retrieval, LSH is widely adopted in many domains, e.g., database, image retrieval, and clustering. Many LSH approaches have been proposed for different purposes. These LSH approaches can be classified into metric-driven and data-driven approaches (Ioffe, 2010). The former approaches aim at approximating specific distance metrics, e.g., cosine distance, Euclidean distance, and Jaccard distance. The latter approaches aim at learning hash functions from the training data to gain optimal performance on specified tasks.

MinHash is designed to approximate the Jaccard distance, which is usually used to measure the similarity between two finite sets. Assuming that A and B are two sets, the Jaccard distance between A and B is defined by

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

Given a set E, and a series of random hash functions $f_i : E \to Z$ mapping each element in E to a distinct integer, let $A \subseteq E$, $B \subseteq E$. The MinHash value of A under the order inferred by f_i is calculated by

$$h(A, f_i) = \underset{x \in A}{argminf_i}(x)) \tag{2}$$

It has been proved (Broder et al., 2000) that if f_i is uniformly and randomly chosen,

$$Pr(h(A,f_i) = h(B,f_i)) = \frac{|A \cap B|}{|A \cup B|} = J(A,B) \tag{3}$$

Method design

In this section, we discuss *NAPIRec* in detail. Generally, our approach seamlessly integrates three phases: the *privacy preservation phase, prediction phase*, and *recommendation phase*. Concretely, in the *privacy preservation phase, NAPIRec* adopts the MinHash technique to encode the API usage data of each enterprise application into several binary bits. In the *prediction phase*, a collaborative filtering-based algorithm is proposed to predict the consumption possibilities of unused APIs for each enterprise application. In the *recommendation phase, NAPIRec* reweights the usage probability of APIs based on the

popularity of APIs and returns a certain number of APIs with the highest possibilities to enterprise applications.

Privacy preservation phase

Step 1: Encode API usage data employing MinHash to preserve the privacy of enterprises. For the usage matrix M, the ith row represents the API usage data of the ith enterprise application. Given a family $\mathscr{F}$ of functions, $I = \{0, 1, \ldots, n-1\}$, for each function $f(\cdot) \in \mathscr{F} : I \to I$, $f(\cdot)$ maps I into a random permutation of elements in the set I. The form of $f(\cdot)$ is defined by

$$f(i) = (d \times i + 1) \ mod \ n \tag{4}$$

where d is a random positive integer and i is the index of an API. Moreover, to satisfy $\forall j \in I, j \neq i, f(i) \neq f(j)$, d and n must be relatively prime. The family $\mathscr{H}$ of hash functions is a set of functions $h : E \to Z^+$. Letting u_i denotes the API usage data of the ith enterprise, i.e., the ith row of usage matrix M. The MinHash value of u_i under the order induced by f is defined as follows:

$$h(u_i, f) = \min_{j \in S_i} f(j) \tag{5}$$

where $S_i = \{j | u_{i,j} = 1\}$.

Prediction phase

Step 2: Calculate the similarity between two enterprises according to hash values. In Step 1, under the order inferred by each $f \in \mathscr{F}$, the API usage data of each enterprise are hashed to an integer value. Consequently, all the hash values of u_i are represented as

$$H(u_i) = \{h(u_i, f_1), h(u_i, f_2), \ldots, h(u_i, f_z)\} \tag{6}$$

The similarity between u_i and u_j is calculated by

$$h(u_i, f_s) \ominus h(u_j, f_s) = \begin{cases} 1, & h(u_i, f_s) = h(u_j, f_s) \\ 0, & h(u_i, f_s) \neq h(u_j, f_s) \end{cases} \tag{7}$$

$$sim(u_i, u_j) = \sum_{k=1}^{z} \left(h(u_i, f_k) \ominus h(u_j, f_k)\right) \tag{8}$$

Step 3: Predict consumption possibilities for unused APIs. In this step, for each enterprise, we select the k most similar enterprises into a set, i,e, *NB_Set*, and then predict the consumption possibility for an unused API by

$$u_{i,j} = \frac{\sum_{t \in NB\_Set} u_{t,j} \omega_t}{k} \tag{9}$$

Recommendation phase

Step 4: Reweight predicted values and make top-k recommendations for enterprise applications. As mentioned in Section "Popularity bias in API recommendation", traditional CF-based recommendation approaches may suffer from popularity bias issues. To alleviate the popularity bias in the recommendation list, we assign different weights to APIs according to their popularity and the length of the recommendation list, i.e., r. The weight of API a_i is calculated as follows:

$$weight = \left(\frac{r}{log(p_i) + 1}\right)^{\alpha} \tag{10}$$

where p_i denotes the popularity of a_i and α is a hyperparameter controlling the influence of popularity in API recommendation. Then we multiply the predicted values of unused APIs by their weights and select the K APIs with highest values as the final recommendation results.

Results and discussion

In this section, we describe experimental settings and then show the advantages of *NAPIRec* based on the following research questions:

- RQ1: Does *NAPIRec* outperform state-of-the-art API recommendation approaches in accuracy and novelty?
- RQ2: How does the size of function family $\mathscr{F}$ and the number of most similar enterprise applications, i.e., k, affect the performance of *NAPIRec*?
- RQ3: How does the hyperparameter α affect the novelty of the recommendation results provided by *NAPIRec*?

Experimental setting

Dataset. To simulate the scenario of API recommendation for enterprises, we collected 18,748 APIs and 6146 mashups from PW. Each mashup is an enterprise application. We removed the mashups that invoked less than two APIs. To validate the performance of *NAPIRec* in recommending less popular APIs for enterprises, we removed the API with the lowest popularity in each mashup. The ground truth of the experiments is that these removed APIs are most beneficial to corresponding enterprise applications.

Experimental Environment. All experiments are performed on a machine with Intel i9-10,900K CPU 3.70 GHz, 32 GB RAM, running Windows 10 × 64 Professional and Python 3.8.

Performance Metrics. We adopt four commonly employed metrics to comprehensively measure the performance of *NAPIRec* from two perspectives: accuracy and novelty. These metrics are within the range of [0, 1].

- **Mean Precision (MP).** Given an API recommendation list, the precision is the ratio of correctly recommended APIs to all the APIs in the list (Tang et al., 2021). MP averages precisions across all recommendation lists. A more significant value indicates better accuracy.
- **Mean Recall (MR).** Given an API recommendation list, recall is considered the ratio of correctly recommended APIs to all APIs that should be recommended (Ma et al., 2021). MR averages recall across all recommendation lists. A more significant value is better.
- **Normalized discounted cumulative gain (NDCG).** NDCG measures the ranking quality of recommendation results and is calculated by

$$NDCG@K = \frac{1}{R} \sum_{i=1}^{K} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{11}$$

where K is the length of the recommendation list and R represents the ideal discounted cumulative gain. A larger NDCG value indicates better performance.

- **Coverage (Cov).** Coverage indicates the fraction of distinct APIs in all recommendation lists to all the APIs in set A (Zhou et al., 2021). Assume that the recommendation lists for all the enterprises are represented as the set $RL = \{rl_1, rl_2, \ldots, rl_m\}$. The Cov value is calculated as follows:

$$COV = \frac{|rl_1 \cup rl_2 \cup \ldots \cup rl_m|}{|A|} \tag{12}$$

A higher COV value means that the recommendation method recommends more novel APIs to enterprises. If a recommendation approach suffers more from popularity bias, the COV value will be low.

Compared Methods. In the experiments, to testify to the effectiveness of our proposed approach in privacy-preserving and

Table 2
Performance Comparison across different K.

Methods	$K = 1$				$K = 5$			
	MP	MR	NDCG	COV	MP	MR	NDCG	COV
POP	0.02	0.02	0.02	0.01	0.04	0.18	0.11	0.02
UPCC	0.20	0.20	0.20	0.15	0.06	0.30	0.25	0.36
UJACC	0.18	0.18	0.18	0.11	0.06	0.30	0.25	0.28
MF	0.00	0.00	0.00	**0.30**	0.03	0.16	0.08	**0.52**
LSH	0.11	0.11	0.11	0.06	0.04	0.21	0.16	0.18
APIRec	0.19	0.19	0.19	0.11	0.06	0.31	0.25	0.28
NAPIRec	**0.21**	**0.21**	**0.21**	0.21	**0.07**	**0.33**	**0.27**	0.45
Methods	$K = 10$				$K = 15$			
	MP	MR	NDCG	COV	MP	MR	NDCG	COV
POP	0.02	0.23	0.12	0.03	0.02	0.28	0.14	0.04
UPCC	0.04	0.40	0.28	0.46	0.03	0.44	0.29	0.50
UJACC	0.04	0.39	0.27	0.40	0.03	0.45	0.29	0.45
MF	0.02	0.23	0.10	**0.56**	0.02	0.27	0.12	**0.57**
LSH	0.03	0.31	0.19	0.33	0.02	0.34	0.20	0.44
APIRec	0.04	0.40	0.28	0.39	0.03	0.45	0.29	0.46
NAPIRec	**0.04**	**0.41**	**0.30**	0.51	**0.03**	**0.46**	**0.31**	0.52

recommending novelty APIs, we compare *NAPIRec* with six classical approaches:

- ***POP***. The popularity-based method always recommends the most popular APIs that have not been consumed by enterprises, which is the baseline method in our experiments.
- ***UPCC***, ***UJACC***. User-based collaborative filtering algorithms adopt Pearson and Jaccard coefficients as similarity metrics. The two methods are also baselines in our experiments.
- ***MF***. (Koren et al., 2009) Matrix factorization decomposes the usage matrix into latent factors to obtain better performance.
- ***SerRec$_{distri_LSH}$***. (Qi et al., 2018) Locality-sensitive hashing-based collaborative filtering approach has the ability to preserve data privacy.
- ***ARIRec***. API recommendations can preserve data privacy but do not reweight the predicted values.

To conduct a fair evaluation, we tune the parameters of each approach to achieve the best performance. In detail, we set the number of nearest neighbors $k = 600$ for *UPCC* and *UJACC*. For *SerRec$_{distri_LSH}$*, the number of hash functions and tables is set to 8. We set the dimension of latent factors to 604 for the *MF* approach.

RQ1: Performance comparison

Table 2 compares the performance of *NAPIRec* with six competitive approaches. The comparison demonstrates that *NAPIRec* outperforms all competing methods in terms of accuracy and novelty, as indicated by four metrics.

Specifically, *NAPIRec* significantly improved over *POP* by 240.01%, 240.01%, 286.83%, and 1580.01% in terms of MP, MR, NDCG, and COV, respectively; it also significantly improved over *MF* by 86.35%, 86.35%, 192.36%, −15.31%. It should be noted that the improvement of *MF* in COV is achieved at the cost of a decrease in accuracy.

The ways of predicting *UPCC*, *UJACC* and *APIRec* are similar to *NAPIRec*. We discover that *NAPIRec* shows a slight advantage over *UPCC*, *UJACC*, and *APIRec* in MP, MR, and NDCG. However, *NAPIRec* achieves better improvements over the three approaches in COV, i.e., 18.98%, 49.91%, and 50.01%, which fully demonstrates the effectiveness of *NAPIRec* in improving the novelty of recommendation results.

Moreover, as two recommendation approaches capable of preserving the privacy of enterprises, *NAPIRec* demonstrates significant advantages over *SerRec$_{distri_LSH}$* in MP, MR, NDCG, and COV, i.e., 56.14%, 56.14%, 70.64%, and 112.6%, respectively. These results show the effectiveness of *NAPIRec* in balancing privacy preservation, accuracy, and novelty.

RQ2: Performance analysis w.r.t z and k

Impact of z. Parameter z indicates the number of functions in $\mathscr{F}$, which is equal to the number of hash tables in MinHash. Parameter z determines the accuracy of MinHash in identifying approximate nearest neighbors. To investigate its impact on the performance of *APIRec* and *NAPIRec*, we vary z from 60 to 270 in steps of 30. Figure 7 shows the performance of *APIRec* in the first row and *NAPIRec* in the second row.

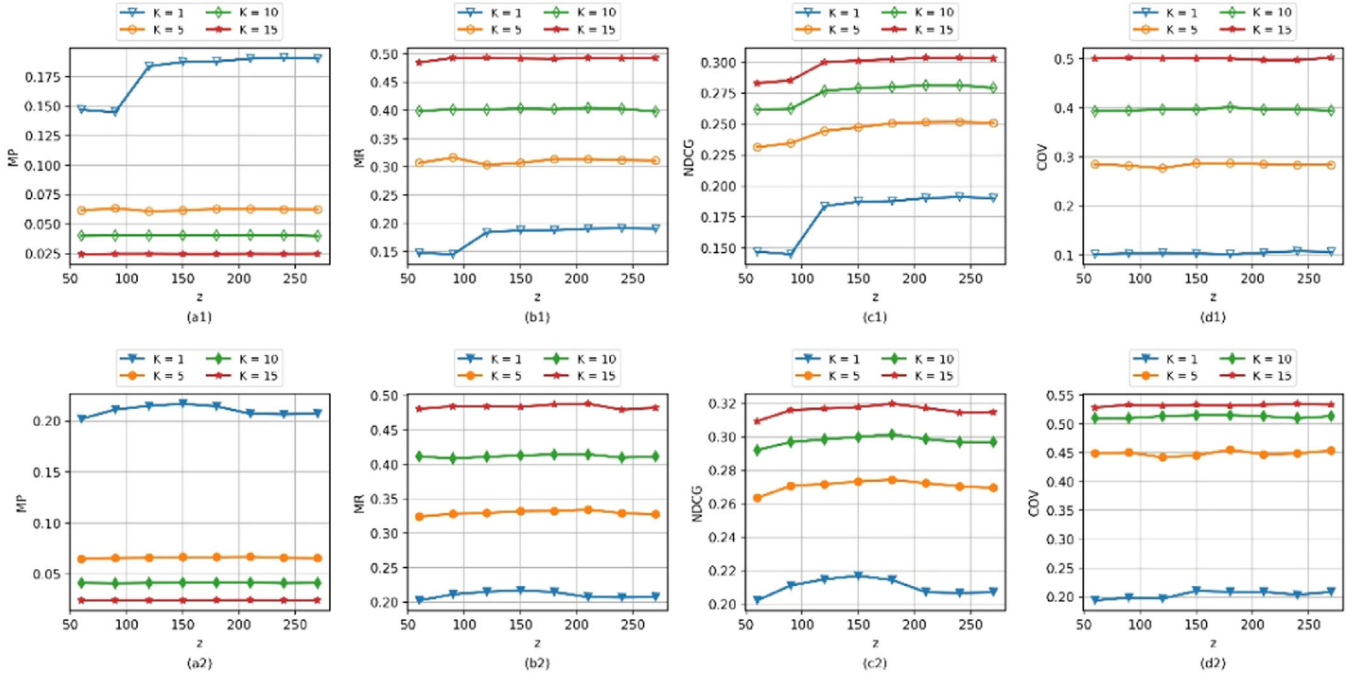


Fig. 7. Impact of z on the performance of *APIRec* and *NAPIRec*.

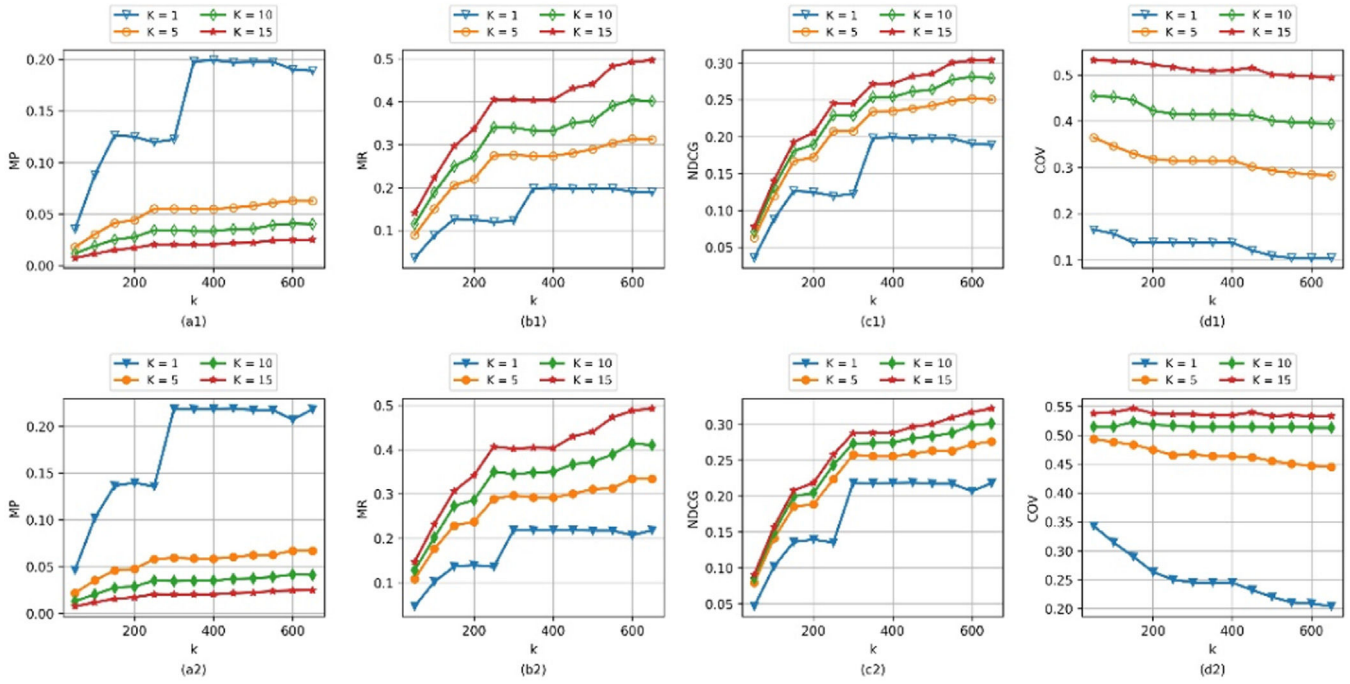


Fig. 8. Impact of k on the performance of *APIRec* and *NAPIRec*.

We observe that as z increases from 50 to 150, the performance of *APIRec* shows a significant increase in MP, MR, and NDCG. When $z >$ 180, the increase in z does not significantly increase the performance of *APIRec*. In addition, the increase in z slightly impacts the performance of *NAPIRec* in MP, MR, and COV. Moreover, when $z > 180$, the NDCG value of *NAPIRec* drops slightly. Therefore, we choose 180 as a reasonable value for z.

Impact of k. *APIRec* and *NAPIRec* utilize information from similar enterprise applications to predict the consumption possibilities of APIs for an enterprise application. Parameter k indicates how many similar enterprise applications are employed in making predictions. To evaluate the impact of k, we vary k from 50 to 650 in steps of 50. Figure 8 demonstrates the influence on the performance of *APIRec* in the first row and *NAPIRec* in the second row. We discover that *APIRec* and *NAPIRec* significantly improved in MP, MR, and NDCG as k increased. In contrast, the COV values of the two approaches decrease slightly as k increases. Thus, the appropriate value of k is set to 600.

RQ3: Performance analysis w.r.t. α

Parameter α is used to set the weight of APIs. It controls the degree of reliance on popularity in API recommendation. An overly small α, i.e., $\alpha = 0$, may lead to a loss in novelty. On the other hand, an excessively large α disregards the positive effect of popularity bias, which may result in a loss in accuracy. To investigate the impact of α, we vary α from 0 to 1.8 in steps of 0.3. The statistical results are shown in Fig. 9. We can find that when $\alpha <$ 1.5, the MP and MR values slightly change as α increases. The NDCG and COV values significantly improve as α increases from 0 to 1.5. However, when $\alpha > 1.5$, the performance of *NAPIRec* decreases in MP, MR, NDCG and slightly increases in COV. Therefore, an appropriate value of α is identified, i.e., $\alpha = 1.5$.

Discussion

There are a few external and internal validity threats in our evaluation. One of the main potential threats to the external validity in evaluation is the representativeness of our dataset. To mitigate this threat, we crawl mashup and API data from PW, one of the world's most famous and largest API repositories. An increasing number of APIs and mashups are registering in PW to further improve its representativeness. Another potential threat is whether our experiments can reflect real-world recommendation demands. To minimize this threat, we simulate the to-be-improved version of these mashups by

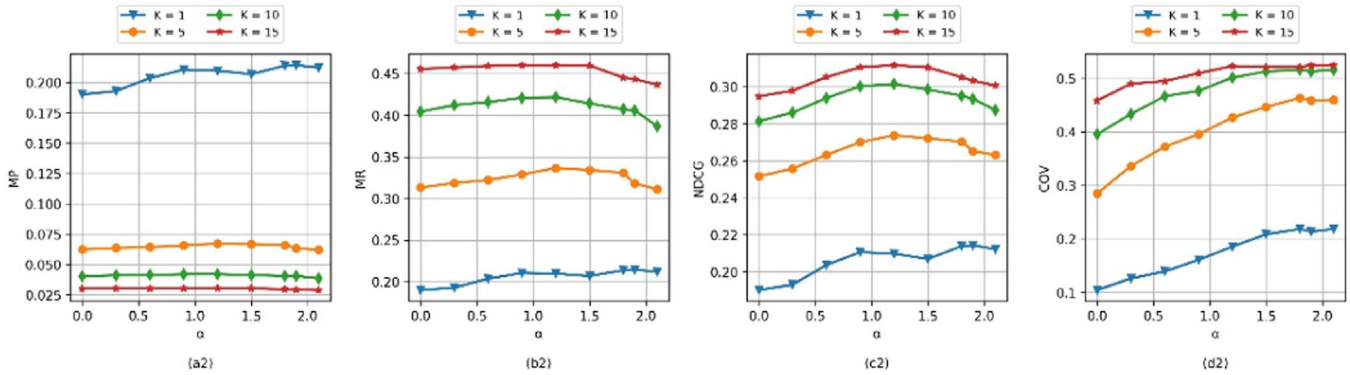


Fig. 9. Impact of α on the performance of *NAPIRec*.

removing APIs that are currently invoked by mashups. These removed APIs are the ground truth for evaluation and can adequately validate the capability of our approach in making accurate recommendations for enterprises.

Our internal validity is whether the comparison with *POP, UPCC, UJACC, MF,* and $SerRec_{distri\_LSH}$ correctly verifies the performance of our proposed approach. We mitigate this threat in four different ways. First, we implement a popularity-based method, i.e., *POP*, as a baseline to analyze the novelty of recommendation results in experiments. Second, to demonstrate the accuracy of our proposed approach, we compare *NAPIRec* with state-of-the-art collaborative filtering approaches. *UPCC* and *UJACC* are two representatives of user-based collaborative filtering methods. *MF* is one of the most successful collaborative filtering recommendation methods. Third, we also compare *NAPIRec* with $SerRec_{distri\_LSH}$, which considers preserving the privacy of service providers while making recommendations. Fourth, to demonstrate the validity of our proposed approach in mitigating popularity bias, we compare *NAPIRec* with *APIRec*, which adopts the same approach as *NAPIRec* without reweighting predicted values. Moreover, we compare these approaches in terms of not only novelty but also accuracy.

Conclusion

In this paper, first, we reveal the importance of API recommendation in facilitating innovation in enterprises. Second, we point out two challenges raised in current API recommendation approaches. To address the first challenge, we introduce MinHash into a classical collaborative filtering approach to preserve data privacy in API recommendations. To address the second challenge, we investigate the popularity bias in a collaborative filtering recommendation approach, and propose a reweighting mechanism to mitigate the popularity bias. Comprehensive experiments are conducted on a real-world dataset obtained from PW. Experimental results show the superiority of *NAPIRec* in preserving data privacy and mitigating popularity bias.

There are still limitations to our proposed method. First, as an implicit feedback-based approach, *NAPIRec* provides enterprises with a suggestion list without knowing the specific requirements of enterprises. Therefore, the accuracy is lower than that of keyword search-based approaches. Second, because *NAPIRec* makes recommendations utilizing historical usage information, it may suffer from cold-start problems. Third, because *NAPIRec* assigns higher priorities to less popular APIs, it may exclude some potential APIs with high popularity from the recommendation list. Thus, it may not be suitable in some cases, e.g., enterprises taking the API adoption rate as the highest priority. However, in most cases, recommending popular APIs is trivial for enterprises.

In future work, we will collect more API usage data from other sources to validate the effectiveness of *NAPIRec*. Moreover, we plan to leverage additional information of APIs, e.g., category and description, to further improve the performance of *NAPIRec*.

Statements and declarations

Consent

All authors have approved the manuscript.

Data availability

Data are available on request due to privacy or other restrictions.

Declaration of Competing Interest

The authors have no relevant financial or non-financial interests to disclose.

Funding

This work was supported by the National Planning Office of Philosophy and Social Science of China (Grant No. 21BJY206).

References

Adomavicius, G., & Zhang, J. (2012). Impact of data characteristics on recommender systems performance. *ACM Transactions on Management Information Systems, 3*(1), 1–17.

Bai, B., Fan, Y., Tan, W., & Zhang, J. (2020). DLTSR: A deep learning framework for recommendations of long-tail web services. *IEEE Transactions on Services Computing, 13*(1), 73–85. doi:10.1109/TSC.2017.2681666.

Bonardi, M., Brioschi, M., Fuggetta, A., Verga, E. S., & Zuccalà, M. (2016). Fostering collaboration through API economy. *IEEE/ACM 3rd International workshop on software engineering research and industrial practice (SER&IP)* (pp. 32–38). doi:10.1145/2897022.2897026.

Broder, A. Z., Charikar, M., Frieze, A. M., & Mitzenmacher, M. (2000). Min-wise independent permutations. *Journal of Computer and System Sciences, 60*(3), 630–659.

Calero, C., Mancebo, J., García, F., Moraga, M.Á., Berná, J. A. G., Fernández-Alemán, J. L., et al. (2019). 5Ws of green and sustainable software. *Tsinghua Science and Technology, 25*(3), 401–414.

Catlett, C., Beckman, P., Ferrier, N., Nusbaum, H., Papka, M. E., Berman, M. G., et al. (2020). Measuring cities with software-defined sensors. *Journal of Social Computing, 1*(1), 14–27.

Cheng, H., Zhong, M., & Wang, J. (2020). Diversified keyword search based web service composition. *Journal of Systems and Software, 163*, 110540. doi:10.1016/j.jss.2020.110540.

Evans, P. C., & Basole, R. C. (2016). Economic and business dimensions: Revealing the API ecosystem and enterprise strategy via visual analytics. *Communications of the ACM, 59*(2), 26–28. doi:10.1145/2856447.

Gao, W., & Wu, J. (2017). A novel framework for service set recommendation in mashup creation. *2017 IEEE International conference on web services (ICWS)* (pp. 65–72). doi:10.1109/ICWS.2017.17.

Gionis, A., Indyk, P., & Motwani, R. (1999). Similarity search in high dimensions via hashing. *25th International conference on very large data bases* (pp. 518–529).

Gong, W., Lv, C., Duan, Y., Liu, Z., Khosravi, M. R., Qi, L., et al. (2021). Keywords-driven web APIs group recommendation for automatic app service creation process. *Software: Practice and Experience, 51*(11), 2337–2354. doi:10.1002/spe.2902.

He, Q., Li, B., Chen, F., Grundy, J., Xia, X., & Yang, Y. (2022). Diversified third-party library prediction for mobile app development. *IEEE Transactions on Software Engineering, 48*(1), 150–165. doi:10.1109/TSE.2020.2982154.

Hu, X., Xiang, Y., Li, Y., Qiu, B., Wang, K., & Li, J. (2021). Trident: Efficient and practical software network monitoring. *Tsinghua Science and Technology, 26*(4), 452–463.

Ioffe, S. (2010). Improved consistent sampling, weighted minhash and L1 sketching. In *Proceedings - IEEE international conference on data mining, ICDM* (pp. 246–255). doi:10.1109/ICDM.2010.80.

Jensen, T. C., & Ashby, D. (2018). *APIs for dummies*. John Wiley & Sons, Inc.

Kang, G., Liu, J., Cao, B., & Cao, M. (2020). NAFM: Neural and attentional factorization machine for web API recommendation. In *Proceedings - 2020 IEEE 13th international conference on web services, ICWS 2020* (pp. 330–337). doi:10.1109/ICWS49710.2020.00050.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer, 42*(8), 30–37.

Liang, Y., Liu, Y., & Gupta, B. B. (2020). PPRP: Preserving-privacy route planning scheme in VANETs. *ACM Transactions on Internet Technology (TOIT)*. doi:10.1145/3430507.

Ma, Y., Sun, H., Chen, Y., Zhang, J., Xu, Y., Wang, X., et al. (2021). DeepPredict: A zone preference prediction system for online lodging platforms. *Journal of Social Computing, 2*(1), 52–70.

Park, Y. J., & Tuzhilin, A. (2008). The long tail of recommender systems and how to leverage it. *ACM Recsys.*

Qi, L., He, Q., Chen, F., Zhang, X., & Ni, Q. (2020). Data-driven web APIs recommendation for building web applications. *IEEE Transactions on Big Data*. doi:10.1109/TBDATA.2020.2975587.

Qi, L., Lin, W., Zhang, X., Dou, W., Xu, X., & Chen, J. (2022). A correlation graph based approach for personalized and compatible web APIs recommendation in mobile APP development. *IEEE Transactions on Knowledge and Data Engineering.*

Qi, L., Zhang, X., Dou, W., Hu, C., Yang, C., & Chen, J. (2018). A two-stage locality-sensitive hashing based approach for privacy-preserving mobile service recommendation in cross-platform edge environment. *Future Generation Computer Systems, 88*, 636–643. doi:10.1016/j.future.2018.02.050.

Sánchez, M. C., de Gea, J. M. C., Fernández-Alemán, J. L., Garcerán, J., & Toval, A. (2019). Software vulnerabilities overview: A descriptive study. *Tsinghua Science and Technology, 25*(2), 270–280.

Tang, S., Huang, S., Zheng, C., Liu, E., Zong, C., & Ding, Y. (2021). A novel cross-project software defect prediction algorithm based on transfer learning. *Tsinghua Science and Technology, 27*(1), 41–57.

Vijayakumar, P., Jegatha, D. L., & Rajkumar, S. C. (2022). Deep reinforcement learning-based pedestrian and independent vehicle safety fortification using intelligent perception. *International Journal of Software Science and Computational Intelligence (IJSSCI), 14*(1), 1–33. doi:10.4018/IJSSCI.291712.

Wang, J., He, D., Castiglione, A., Gupta, B. B., Karuppiah, M., & Wu, L. (2022). PCNNCEC: Efficient and privacy-preserving convolutional neural network inference based on

cloud-edge-client collaboration. *IEEE Transactions on Network Science and Engineering*. doi:10.1109/TNSE.2022.3177755 1–1.

Wang, L., Zhang, X., Wang, T., Wan, S., Srivastava, G., Pang, S., et al. (2021). Diversified and scalable service recommendation with accuracy guarantee. *IEEE Transactions on Computational Social Systems, 8*(5), 1182–1193. doi:10.1109/TCSS.2020.3007812.

Wulf, J., & Blohm, I. (2020). Fostering value creation with digital platforms: A unified theory of the application programming interface design. *Journal of Management Information Systems, 37*(1), 251–281. doi:10.1080/07421222.2019.1705514.

Xiao, Y., Liu, J., Kang, G., Hu, R., Cao, B., Cao, Y., et al. (2020). Structure reinforcing and attribute weakening network based API recommendation approach for mashup creation. *2020 IEEE International conference on web services (ICWS)* (pp. 541–548). doi:10.1109/ICWS49710.2020.00078.

Yan, R., Fan, Y., Zhang, J., Zhang, J., & Lin, H. (2021). Service recommendation for composition creation based on collaborative attention convolutional network. *2021 IEEE International conference on web services (ICWS)* (pp. 397–405). doi:10.1109/ICWS53863.2021.00059.

Yang, H., Vijayakumar, P., Shen, J., & Gupta, B. B. (2022). A location-based privacy-preserving oblivious sharing scheme for indoor navigation. *Future Generation Computer Systems, 137*, 42–52. doi:10.1016/j.future.2022.06.016.

Yao, L., Wang, X., Sheng, Q. Z., Benatallah, B., & Huang, C. (2021). Mashup recommendation by regularizing matrix factorization with API Co-invocations. *IEEE Transactions on Services Computing, 14*(2), 502–515. doi:10.1109/TSC.2018.2803171.

Zhou, X., Li, Y., & Liang, W. (2020). CNN-RNN based intelligent recommendation for online medical pre-diagnosis support. *IEEE/ACM Transactions on Computational Biology and Bioinformatics, 18*(3), 912–921.

Zhou, X., Xu, X., Liang, W., Zeng, Z., & Yan, Z. (2021). Deep-learning-enhanced multitarget detection for end–edge–cloud surveillance in smart IoT. *IEEE Internet of Things Journal, 8*(16), 12588–12596.