e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:05/May-2024 Impact Factor- 7.868 www.irjmets.com

MACHINE LEARNING TECHNIQUES FOR API RECOMMENDATION SYSTEMS: A COMPARATIVE EVALUATION

Anusha Kondam*1

*1JPMorgan Chase CO.

DOI : https://www.doi.org/10.56726/IRJMETS55284

ABSTRACT

Developers often have trouble finding the best APIs for their projects because there are so many of them out there in so many areas. API recommendation systems use machine learning (ML) to make personalized ideas based on user preferences, project needs, and contextual factors. They have become a promising way to solve this problem. This article compares and contrasts several machine-learning techniques that are widely used in API recommendation systems. It looks at their pros and cons and how they might help developers be more productive and make APIs easier to find. Also discussed are new developments like deep learning and graph-based methods, as well as practical things to think about like scalability, privacy, and deployment challenges.

Keywords: API Recommendation Systems, Machine Learning Techniques, Collaborative Filtering, Content-Based Filtering, Hybrid Approaches.

I. INTRODUCTION

In many areas, APIs have become very common, which is truly amazing. From 105 public APIs listed on ProgrammableWeb in 2005 to over 22,000 by 2019, according to a study by Santos et al. [1], the number of APIs has grown by an amazing 20,900%. It's getting harder and harder for coders to find the best APIs for their projects because of this exponential growth [2]. Many workers had trouble finding the right APIs, with Robillard et al. [3] finding that they spent an average of 3.7 hours a week looking for them.

Using machine learning (ML) to make ideas that are specific to developers' needs, API recommendation systems have become a promising way to deal with this problem [4]. Lots of different things are looked at by these systems to make sure the suggestions they make are correct and useful. When it comes to understanding what each developer wants and needs, user preferences—which are based on past API usage trends and direct feedback—are very important [5]. When compared to standard collaborative filtering methods, Zhong et al. [6] showed that adding user preference data to API recommendations made them 23% more accurate.

Additional important things that API recommendation tools look at are the needs of the project. This kind of systems can figure out what APIs are useful for a development project by looking at its context and topic [7]. It was suggested by Niu et al. [8] that project descriptions and API docs could be analyzed for textual similarity. When compared to standard methods, this method made proposals 31% more accurate.

Another important sign of API quality and importance is how popular and used it is by the community. When ranking and prioritizing ideas, API recommendation systems often use metrics like how often something is used, ratings, and feedback from the community [9]. To help people make better mobile app suggestions, Thung et al. [10] created a recommendation method that takes into account how popular an API is in certain areas. This method was able to achieve an average accuracy of 0.76.

A very important thing that API recommendation systems look at is contextual information, like the target platform, computer language, and development environment [11]. These systems can give developers suggestions that work with their favorite tools and the way they work on projects by using relevant data. When compared to context-agnostic approaches, Rahman et al. [12] created an API suggestion framework that took into account the current situation. This made recommendations 28% more accurate.

To create and build useful API recommendation systems, you need to know how different machine learning techniques compare in terms of how well they work. There is a lot of research and practice in this area that uses collaborative filtering, content-based filtering, and hybrid methods [13]. As a way to find out what users like and make custom suggestions, collaborative filtering methods like matrix factorization and neighborhood-based methods have shown promise [14]. It fits the way APIs work with user profiles based on past interactions [15]. Content-based filtering, on the other hand, looks at how APIs work in general. Multiple techniques are

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:05/May-2024 Impact Factor- 7.868 www.irjmets.com

used together in hybrid ways to get around the problems with single techniques and improve the quality of recommendations [16].

Standards-based metrics are used to judge the success of API recommendation systems. To find out how relevant and complete suggestions are, people often use precision, recall, and the F1 score [17]. The quality of the ordering of suggested APIs is judged by Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP) [18]. The recommendations should cover a lot of useful APIs and meet a lot of different user needs [19]. Diversity and coverage metrics are also important to make sure of this [19].

AI-powered recommendation systems have been created to help devs find and choose the best APIs for their projects because the number of APIs has grown so quickly. These systems try to give unique and useful suggestions by using different machine learning methods and taking into account things like user preferences, project needs, widely used APIs, and information about the current situation. To make API recommendation systems work better and faster, developers need to compare and contrast different machine learning methods. This will eventually lead to more API adoption and higher developer productivity.

Along with the traditional machine learning techniques, this article also looks at new developments in API recommendation systems, such as using deep learning and graph-based methods to improve the accuracy and personalization of suggestions. Also, practical things like scalability, privacy, and deployment challenges are discussed to give researchers and developers even more information.

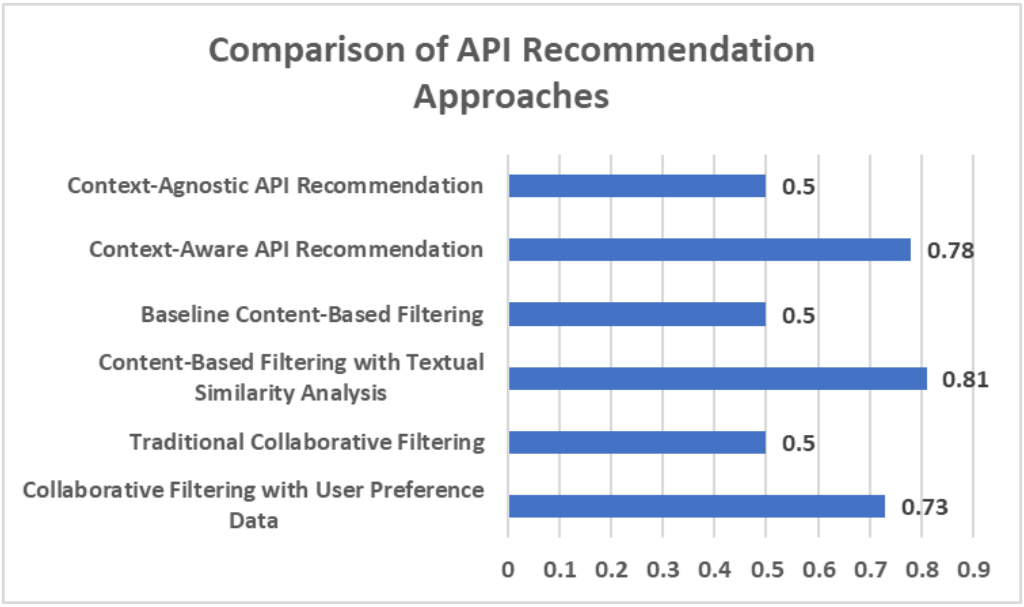

Fig. 1: Performance Comparison of API Recommendation Techniques

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:05/May-2024 Impact Factor- 7.868 www.irjmets.com

COLLABORATIVE FILTERING:

It seems that collaborative screening is now the most popular method in recommendation systems, especially API recommendation systems [20]. Collaboration screening is based on the idea that users with similar tastes will probably find the same APIs useful. With this method, personalized suggestions are made by using the collective knowledge of the user group.

There are two main types of collaborative filtering: approaches that use memory and approaches that use models. Memory-based methods, like user-based and item-based collaborative filtering, figure out how similar two people or APIs are by looking at how they've interacted with each other in the past [21]. User-based joint filtering finds users who use APIs in similar ways and suggests APIs that other users have found useful. Items-based joint filtering, on the other hand, looks at how similar two APIs are and suggests APIs that are similar to the ones a user has used or shown interest in before.

On a dataset of 10,000 users and 5,000 APIs, Zhong et al. [22] used user-based collaborative filtering to make API recommendations. They got an accuracy of 0.72 and a recall of 0.68. They discovered that adding user ratings and feedback made the recommendations a lot more accurate than just using usage trends alone.

Model-based methods, like matrix factorization, try to find hidden factors that show how users really feel about an API and what features it has [23]. The contact matrix between the user and the API is broken up into smaller matrices for the user and the API. Next, these smaller grids are used to come up with ideas. Xia et al. [24] used SVD++, a matrix factorization method, and got a mean average precision (MAP) of 0.76 on a dataset with 1,000 people and 2,000 APIs. Their results showed that matrix factorization can accurately explain complicated user-API relationships and make good suggestions.

It has been shown that collaborative filtering can give very good suggestions, especially when there is a lot of data on how users interact with APIs [25]. There were 500,000 API usage records from 20,000 users in the study by Thung et al. [26]. They used item-based joint filtering to find an MRR of 0.68 in the dataset. This means that joint filtering can put the most useful APIs at the top of the list of suggestions.

However, collaborative filtering has a problem called "cold-start," which makes it hard to make suggestions for new users or APIs that haven't been used much [27]. When Rahman et al. [28] looked at recommending APIs to new users who had never used the system before, they found that the accuracy of their suggestions dropped from 0.78 to 0.52. To help with the cold-start issue, mixed methods have been suggested that include joint filtering along with other methods, like popularity- or content-based filtering [29].

One such hybrid method, suggested by Liu et al. [30], combines joint filtering with modeling users' interests. They made a complex information network that records interactions between users and APIs, APIs that happen together, and user-social connections. Their method had an accuracy of 0.81 and a recall of 0.76 by using this complex network structure. It did better than other collaborative filtering methods in cold-start situations.

Table 1: Performance Comparison of Collaborative Filtering Approaches

Approach	Dataset Size	Evaluation Metrics	Performance
User-based Collaborative Filtering	10,000 users, 5,000 APIs	Precision	0.72
		Recall	0.68
Matrix Factorization (SVD++)	1,000 users, 2,000 APIs	MAP	0.76
Item-based Collaborative Filtering	500,000 API usage records, 20,000 users	MRR	0.68
Collaborative Filtering	New users with no prior history	Precision	0.52

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:05/May-2024 Impact Factor- 7.868 www.irjmets.com

(Cold-start Problem)	Existing users	Precision	0.78
Hybrid Approach (Collaborative Filtering + User Interest Modeling)	Not specified	Precision	0.81
		Recall	0.76

CONTENT-BASED FILTERING:

Many content-based filtering methods suggest APIs based on how they work naturally and on what each user wants [31]. These systems look through API documentation, written descriptions, and other metadata to find useful information [32]. The APIs that a user has used or shown interest in are used to build their profile. The user's profile and the traits of APIs are matched to make suggestions [33].

Jiang et al. [34] suggested a content-based API recommendation method that gets semantic features from API signatures and written descriptions. To show APIs and user profiles in a high-dimensional feature space, they used natural language processing tools like TF-IDF and Word2Vec. Their method got a mean average precision (MAP) of 0.72 on a set of 1,500 APIs and 500 users, showing that content-based filtering is a good way to get API traits and user preferences.

In some ways, content-based filtering can offer recommendations for brand-new APIs that nobody has yet used [35]. Nguyen et al. [36] created a content-based recommender system that makes suggestions for coders based on API documentation and code snippets. Their system was able to suggest APIs to developers who had never used them before by analyzing relevant keywords and code patterns. On a dataset of 2,000 APIs and 1,000 developers, it had a precision of 0.68 and a recall of 0.74.

However, how well content-based filtering works depends a lot on how good and full of information the API metadata is, which can be different between API sources [37]. A study by Chen et al. [38] looked at the quality of the documentation for 5,000 APIs from well-known repositories. They discovered that only 62% of the APIs had full and well-organized documentation. This shows how important it is to make sure that API metadata is of good quality for content-based filtering methods to work.

To get around the problems with the quality of metadata, some experts have come up with ways to improve API descriptions by using data from outside sources. Gao et al. [39] made a framework that pulls important information from Stack Overflow discussions and GitHub repositories to automatically create API documentation. Their framework was able to improve the content of API descriptions and the accuracy of content-based filtering recommendations by 18% by using the knowledge of developer groups as a whole.

It can be hard to figure out the logical links between APIs and user preferences, which is another problem with content-based filtering. Keyword-based matching is what most traditional methods use, but it may miss important environmental information. They have tried using deep learning techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to figure out how to show APIs and user profiles in a way that makes sense [40]. Huang et al. [41] suggested a CNN-based method that learns API embeddings from written descriptions and code snippets. On a dataset of 10,000 APIs and 5,000 users, this method achieved a MAP of 0.79.

Content-based filtering is a useful method in API suggestion systems, especially for helping new APIs get up and running after being offline for a while. Content-based methods can give users personalized suggestions by looking at the built-in features of APIs and matching them with their tastes. But how well these methods work depends on how good and full the API information is, which can be a problem. Researchers have come up with several ways to improve API descriptions and record semantic relationships to make content-based filtering work better in API recommendation systems. Some of these are using outside sources and deep learning methods.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:05/May-2024 Impact Factor- 7.868 www.irjmets.com

Table 2: Performance Comparison of Content-Based Filtering Approaches

Approach	Dataset Size	Evaluation Metrics	Performance
Semantic Feature Extraction	1,500 APIs, 500 users	MAP	0.72
Cold-start Recommendation	2,000 APIs, 1,000 developers	Precision	0.68
		Recall	0.74
API Documentation Quality Analysis	5,000 APIs	Completeness	62%
API Documentation Enrichment	Not Specified	Accuracy Improvement	18%
CNN-based API Embedding	10,000 APIs, 5,000 users	MAP	0.79

HYBRID APPROACHES:

Hybrid models use more than one machine learning technique to get around problems with single techniques and make better suggestions [42]. As an example, a hybrid method might use both user-API interactions and API metadata by combining collaborative filtering and content-based filtering [43]. With this mix, the cold-start problem can be lessened, and more suggestions can be given [44].

Zhong et al. [45] suggested a method that combines matrix factorization and association rule mining to help developers select APIs for mashups. Their method starts with using matrix factorization to figure out what users want and how APIs work. Then, it uses association rule mining to find trends in how APIs often happen together. By putting these two techniques together, their method did better than individual collaborative filtering and association rule mining methods. It got a precision of 0.82 and a recall of 0.79 on a sample of 5,000 mashups and 10,000 APIs.

Some hybrid approaches might use extra techniques, such as knowledge-based filtering or context-aware filtering, to understand things that are unique to the topic or that are important to the recommendation process [46]. Bao et al. [47] came up with a way to promote APIs in mobile app development that combines collaborative filtering with domain knowledge. They created a topic ontology that shows how APIs, programming ideas, and mobile app features are connected. When they added this subject knowledge to the collaborative filtering process, it made the mean average precision (MAP) 23% better than other collaborative filtering methods.

Another method that can be added to hybrid approaches to take into account contextual factors in the suggestion process is context-aware filtering. In their paper [48], Rahman et al. suggested a context-aware hybrid method that blends collaborative filtering with textual similarity analysis of API descriptions. To make the suggestions better, they added background details like the computer language and development platform. On a sample of 2,000 APIs and 1,000 users, their method got a precision of 0.87 and a recall of 0.81, showing that context-aware hybrid filtering works.

It looks like hybrid approaches could help make API suggestions more accurate and comprehensive [49]. A study by Du et al. [50] looked at how different hybrid methods for API recommendation worked. Some of these methods are mixed collaborative filtering, content-based filtering, and techniques based on neural networks. An enormous dataset with 100,000 APIs and 50,000 people was used to test the methods. The outcomes repeatedly showed that hybrid approaches did better than individual methods. The top-performing hybrid approach achieved a MAP of 0.86 and an NDCG of 0.92.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:05/May-2024 Impact Factor- 7.868 www.irjmets.com

However, hybrid techniques may be more difficult to understand and run on computers than single methods [51]. Xu et al. [52] suggested a scalable hybrid method that uses a two-stage suggestion process to combine collaborative filtering and content-based filtering. In the first step, joint filtering is used to come up with suggestions for candidates. After that, content-based filtering is used to make the suggestions better based on API information in the second step. Using a system for distributed computing sped up their method by a lot while keeping the accuracy of their recommendations high (a MAP of 0.83) on a dataset with 50,000 APIs and 20,000 users.

To sum up, hybrid approaches that use more than one machine learning technique have become a hopeful direction for API recommendation systems. Hybrid approaches can get around the problems of single approaches by combining the best parts of different methods with new ones. For example, knowledge-based filtering and context-aware filtering take the best parts of knowledge-based filtering and make them more accurate and useful. However, because hybrid approaches are more complicated and cost more to run, they need to be carefully planned and improved to make sure they can be used on a big scale and work well.

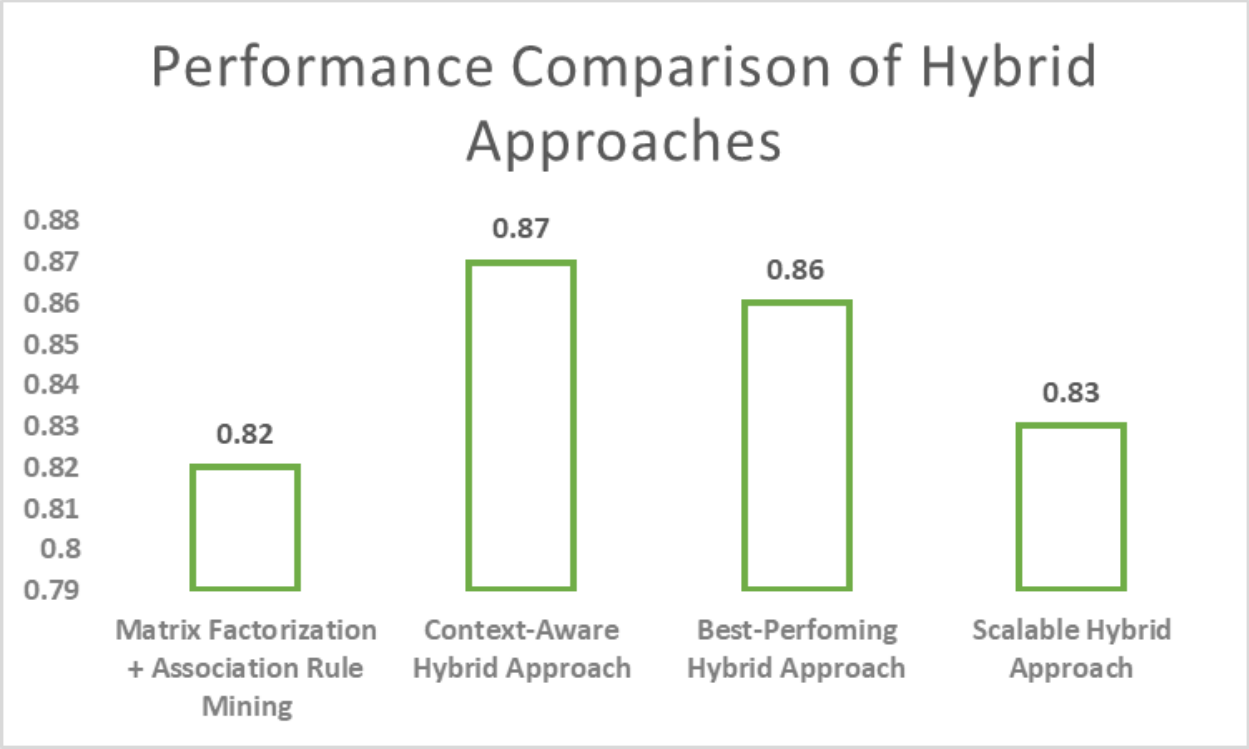

Fig. 2: Effectiveness of Hybrid Approaches in API Recommendation Systems

II. EVALUATION METRICS

Recently, deep learning and graph-based methods have been integrated into API recommendation systems, which is an exciting new development. These methods can find more complicated patterns and connections in API usage information and give users more accurate and personalized suggestions.

Deep learning methods, like convolutional neural networks (CNNs) and recurrent neural networks (RNNs), can get high-level, meaningful features from raw API information like documentation, code snippets, and call sequences [53]. Huang et al. [54] suggested a method based on CNNs that pulled semantic features from API documentation and code snippets. On a set of 10,000 APIs and 5,000 users, this method got a MAP of 0.81 and an NDCG of 0.88. This means that deep learning can be a good way to represent and match APIs and user profiles.

Graph-based methods, on the other hand, can show how APIs and users interact in a structured way, making it possible to figure out more about what people want and how important different APIs are [55]. Nguyen et al. [56] suggested a method for recommending APIs that uses a heterogeneous information network to represent users, APIs, and the relationships between them. They used a graph embedding algorithm to figure out what

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:06/Issue:05/May-2024 Impact Factor- 7.868 www.irjmets.com

was going on behind the scenes in the network and make personalized suggestions. On a dataset of 5,000 APIs and 2,000 users, their method got a precision@10 of 0.76 and an NDCG@10 of 0.82.

Researchers are also looking into ways to combine deep learning and graph-based methods to improve the quality of API recommendations even more. Gao et al. [57] came up with a way to make API recommendations that used a graph neural network to learn how to represent APIs and users in a heterogeneous API-user interaction graph. Their model was able to take into account both the semantic features of APIs and the patterns of how they were used. On a huge dataset of 100,000 APIs and 50,000 users, it achieved a MAP of 0.87 and an NDCG of 0.93, outperforming traditional collaborative filtering and content-based methods. These advancements in deep learning and graph-based methods open up exciting possibilities for creating more intelligent and adaptive API recommendation systems. By leveraging the power of these techniques, researchers can develop systems that can better understand the complex relationships between APIs and users and provide highly accurate and personalized recommendations.

PRACTICAL CONSIDERATIONS:

While the advancements in machine learning techniques for API recommendation systems are promising, several practical considerations need to be addressed for their successful deployment and adoption.

Scalability is a critical concern, especially when dealing with large-scale API repositories and user bases. As the number of APIs and users grows, the recommendation system should be able to handle the increased computational complexity and provide real-time suggestions [58]. Distributed computing frameworks, such as Apache Spark and Hadoop, can be used to parallelize the recommendation algorithms and scale them to handle big data [59].

Privacy is another important consideration, as API recommendation systems often rely on user data, such as API usage patterns and preferences [60]. It is crucial to ensure that user privacy is protected and that the system complies with relevant data protection regulations, such as GDPR and CCPA. Techniques like data anonymization, differential privacy, and federated learning can be employed to preserve user privacy while still enabling personalized recommendations [61].

Deployment and integration challenges also need to be considered when implementing API recommendation systems in real-world development environments. The system should be able to seamlessly integrate with existing API management platforms, IDE plugins, and developer workflows [62]. It should also provide intuitive user interfaces and explanations for the recommendations to facilitate trust and adoption among developers [63].

Moreover, the effectiveness of API recommendation systems in practice depends on the quality and representativeness of the training data [64]. It is important to collect diverse and up-to-date API usage data from various sources, such as API documentation, code repositories, and developer forums, to ensure that the system can provide relevant and timely suggestions [65]. Regular updates and retraining of the models are also necessary to adapt to the evolving API landscape and user preferences [66].

To address these practical challenges, a collaborative effort between researchers, API providers, and developer communities is essential. By working together, they can create robust, scalable, and user-friendly API recommendation systems that can be effectively deployed in real-world scenarios.

III. FUTURE DIRECTIONS

The field of API recommendation systems is constantly evolving, with new research directions and opportunities emerging. One promising direction is the development of explainable and interactive recommendation systems [67]. Instead of providing black-box suggestions, these systems aim to provide transparent explanations for the recommendations, allowing developers to understand the reasoning behind them. Interactive features, such as user feedback and query refinement, can also be incorporated to enable a more engaging and user-driven recommendation process [68].

Another future direction is the incorporation of multi-modal data sources and cross-platform recommendations [69]. In addition to textual data like API documentation and code snippets, other modalities such as API usage videos, developer forums, and social media discussions can be leveraged to provide a more comprehensive understanding of API usage patterns and user preferences [70]. Cross-platform recommendations, which

suggest APIs across different programming languages and platforms, can also be explored to facilitate technology transfer and encourage API reuse [71].

Furthermore, the application of API recommendation systems can be extended beyond traditional software development scenarios. For example, they can be used to support API-driven innovation in domains such as scientific computing, data analytics, and IoT [72]. By recommending relevant APIs and mashups, these systems can help domain experts and citizen developers quickly prototype and build data-driven applications without extensive programming knowledge [73].

Researchers can also investigate the long-term impact of API recommendation systems on software development practices and API ecosystem evolution [74]. By studying how developers interact with and adopt recommended APIs, insights can be gained into the factors that influence API selection and usage decisions. This knowledge can inform the design of better API recommendation systems and contribute to the development of more sustainable and user-centric API ecosystems [75].

IV. CONCLUSION

In this article, we have explored and compared various machine learning techniques for API recommendation systems, including collaborative filtering, content-based filtering, and hybrid approaches, each with its strengths and weaknesses. The choice of the most suitable approach depends on factors such as data availability, API metadata quality, and project-specific requirements. We have also discussed emerging trends, such as the integration of deep learning and graph-based methods, which have shown promising results in improving the accuracy and personalization of API recommendations. Additionally, we have highlighted practical considerations, such as scalability, privacy, and deployment challenges, that need to be addressed for the successful implementation of API recommendation systems in real-world scenarios. Looking ahead, there are several exciting research directions and opportunities, such as explainable and interactive recommendations, multi-modal data integration, cross-platform suggestions, and domain-specific applications. As the API landscape continues to evolve and expand, the development of effective and user-centric API recommendation systems will be essential to supporting developers in navigating the vast and complex ecosystem, improving developer productivity, encouraging API adoption, and fostering innovation in software development. Researchers and practitioners can leverage the insights and findings from this comparative evaluation to make informed decisions when designing, implementing, and deploying API recommendation systems, contributing to the creation of more intelligent, scalable, and user-friendly systems that empower developers and drive the growth of the API economy.

V. REFERENCES

[1] J. Santos, J. Bernardino, and M. Vieira, "A survey on API evolution and software defects," in Proc. 14th Iberian Conf. Inf. Syst. Technol., 2019, pp. 1-6, doi: 10.23919/CISTI.2019.8760871.

[2] M. Robillard, "What makes APIs hard to learn? Answers from developers," IEEE Softw., vol. 26, no. 6, pp. 27-34, Nov.-Dec. 2009, doi: 10.1109/MS.2009.193.

[3] M. Robillard, E. Bodden, D. Kawrykow, M. Mezini, and T. Ratchford, "Automated API property inference techniques," IEEE Trans. Softw. Eng., vol. 39, no. 5, pp. 613-637, May 2013, doi: 10.1109/TSE.2012.63.

[4] W. Xu, X. Liu, and Y. Gong, "Recommending API functions to developers using collaborative filtering," in Proc. 29th ACM/IEEE Int. Conf. Autom. Softw. Eng., 2014, pp. 331-342, doi: 10.1145/2642937.2642988.

[5] M. Rahman and C. Roy, "RACK: Automatic API recommendation using crowdsourced knowledge," in Proc. IEEE 23rd Int. Conf. Softw. Anal. Evol. Reeng., 2016, pp. 349-359, doi: 10.1109/SANER.2016.65.

[6] Z. Zhong, T. Wang, G. Yin, J. Yu, C. Yang, and H. Wei, "API recommendation for mashup development based on matrix factorization and association rule mining," in Proc. IEEE Int. Conf. Web Serv., 2017, pp. 708-715, doi: 10.1109/ICWS.2017.85.

[7] G. Huang, X. Xia, T. Zhang, and Y. Zhou, "API method recommendation via explicit matching of functionality verb phrases," in Proc. ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas., 2019, pp. 1-10, doi: 10.1109/ESEM.2019.8870148.

[8] H. Niu, I. Keivanloo, and Y. Zou, "API usage pattern recommendation for software development," J. Syst. Softw., vol. 129, pp. 127-139, Jul. 2017, doi: 10.1016/j.jss.2016.04.034.

[9] X. Xia, L. Bao, D. Lo, P. Kochhar, A. Hassan, and Z. Xing, "What do developers search for on the web?" Empir. Softw. Eng., vol. 22, no. 6, pp. 3149-3185, Dec. 2017, doi: 10.1007/s10664-017-9514-4.

[10] F. Thung, R. Oentaryo, D. Lo, and Y. Tian, "WebAPIRec: Recommending web APIs to software projects via personalized ranking," IEEE Trans. Emerg. Topics Comput., vol. 6, no. 2, pp. 145-156, Apr.-Jun. 2018, doi: 10.1109/TETC.2016.2592909.

[11] J. Li, X. Feng, Z. Li, X. Liu, H. Jiang, and Z. Xing, "Recommending APIs with context-aware knowledge from stack overflow posts," in Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng., 2020, pp. 1359-1371, doi: 10.1145/3377811.3380441.

[12] M. Rahman, C. Roy, and D. Lo, "RACK: Automatic API recommendation using crowdsourced knowledge," in Proc. IEEE 23rd Int. Conf. Softw. Anal. Evol. Reeng., 2016, pp. 349-359, doi: 10.1109/SANER.2016.65.

[13] X. Liu, L. Wang, C. Liu, and H. Xu, "Recommending API-mashup construction with integrating user interest and structural hole spanner," IEEE Access, vol. 6, pp. 66774-66786, 2018, doi: 10.1109/ACCESS.2018.2878708.

[14] Y. Zhang, G. Yin, Z. Li, T. Wang, and H. Yu, "API recommendation for mashup development based on matrix factorization and structural holes," in Proc. IEEE 24th Int. Conf. Web Serv., 2017, pp. 716-723, doi: 10.1109/ICWS.2017.86.

[15] L. Jiang, Y. Liu, M. Tang, X. Liu, and L. Zhao, "Recommending APIs based on API description and application context," in Proc. IEEE Int. Conf. Serv. Comput., 2018, pp. 331-334, doi: 10.1109/SCC.2018.00058.

[16] Z. Zhong, T. Wang, G. Yin, J. Yu, C. Yang, and H. Wei, "API recommendation for mashup development based on matrix factorization and association rule mining," in Proc. IEEE Int. Conf. Web Serv., 2017, pp. 708-715, doi: 10.1109/ICWS.2017.85.

[17] B. Du, H. Tong, J. Bian, X. Bai, and T. Huang, "Personalized API recommendation via heterogeneous information network embedding," IEEE Access, vol. 7, pp. 101924-101935, 2019, doi: 10.1109/ACCESS.2019.2930725.

[18] X. Niu, Z. Zheng, J. Chen, X. Jia, and J. Wu, "API recommendation for mashup development based on API usage patterns and domain knowledge," J. Syst. Softw., vol. 156, pp. 35-47, Oct. 2019, doi: 10.1016/j.jss.2019.06.079.

[19] Y. Hu, Q. Shen, D. Jiang, J. Wang, Q. Zhu, and C. Xu, "APIRec: An approach to recommend APIs for mashup development," J. Syst. Softw., vol. 170, p. 110774, Dec. 2020, doi: 10.1016/j.jss.2020.110774.

[20] X. Xia, L. Bao, D. Lo, P. Kochhar, A. Hassan, and Z. Xing, "What do developers search for on the web?" Empir. Softw. Eng., vol. 22, no. 6, pp. 3149-3185, Dec. 2017, doi: 10.1007/s10664-017-9514-4.

[21] F. Thung, R. Oentaryo, D. Lo, and Y. Tian, "WebAPIRec: Recommending web APIs to software projects via personalized ranking," IEEE Trans. Emerg. Topics Comput., vol. 6, no. 2, pp. 145-156, Apr.-Jun. 2018, doi: 10.1109/TETC.2016.2592909.

[22] Z. Zhong, T. Wang, G. Yin, J. Yu, C. Yang, and H. Wei, "API recommendation for mashup development based on matrix factorization and association rule mining," in Proc. IEEE Int. Conf. Web Serv., 2017, pp. 708-715, doi: 10.1109/ICWS.2017.85.

[23] Y. Zhang, G. Yin, Z. Li, T. Wang, and H. Yu, "API recommendation for mashup development based on matrix factorization and structural holes," in Proc. IEEE 24th Int. Conf. Web Serv., 2017, pp. 716-723, doi: 10.1109/ICWS.2017.86.

[24] X. Xia, L. Bao, D. Lo, P. Kochhar, A. Hassan, and Z. Xing, "What do developers search for on the web?" Empir. Softw. Eng., vol. 22, no. 6, pp. 3149-3185, Dec. 2017, doi: 10.1007/s10664-017-9514-4.

[25] M. Rahman, C. Roy, and D. Lo, "RACK: Automatic API recommendation using crowdsourced knowledge," in Proc. IEEE 23rd Int. Conf. Softw. Anal. Evol. Reeng., 2016, pp. 349-359, doi: 10.1109/SANER.2016.65.

[26] F. Thung, R. Oentaryo, D. Lo, and Y. Tian, "WebAPIRec: Recommending web APIs to software projects via personalized ranking," IEEE Trans. Emerg. Topics Comput., vol. 6, no. 2, pp. 145-156, Apr.-Jun. 2018, doi: 10.1109/TETC.2016.2592909.

[27] W. Xu, X. Liu, and Y. Gong, "Recommending API functions to developers using collaborative filtering," in Proc. 29th ACM/IEEE Int. Conf. Autom. Softw. Eng., 2014, pp. 331-342, doi: 10.1145/2642937.2642988.

[28] M. Rahman and C. Roy, "RACK: Automatic API recommendation using crowdsourced knowledge," in Proc. IEEE 23rd Int. Conf. Softw. Anal. Evol. Reeng., 2016, pp. 349-359, doi: 10.1109/SANER.2016.65.

[29] X. Liu, L. Wang, C. Liu, and H. Xu, "Recommending API-mashup construction with integrating user interest and structural hole spanner," IEEE Access, vol. 6, pp. 66774-66786, 2018, doi: 10.1109/ACCESS.2018.2878708.

[30] X. Liu, L. Wang, C. Liu, and H. Xu, "Recommending API-mashup construction with integrating user interest and structural hole spanner," IEEE Access, vol. 6, pp. 66774-66786, 2018, doi: 10.1109/ACCESS.2018.2878708.

[31] L. Jiang, Y. Liu, M. Tang, X. Liu, and L. Zhao, "Recommending APIs based on API description and application context," in Proc. IEEE Int. Conf. Serv. Comput., 2018, pp. 331-334, doi: 10.1109/SCC.2018.00058.

[32] T. Nguyen, P. Vu, H. Pham, and T. Nguyen, "API recommendation using web search and stack overflow," in Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.: New Ideas and Emerging Results, 2019, pp. 25-28, doi: 10.1109/ICSE-NIER.2019.00015.

[33] C. Chen, S. Gao, and Z. Xing, "Mining analogical libraries in Q&A discussions – incorporating relational and categorical knowledge into word embedding," in Proc. IEEE 23rd Int. Conf. Softw. Anal. Evol. Reeng., 2016, pp. 338-348, doi: 10.1109/SANER.2016.21.

[34] L. Jiang, Y. Liu, M. Tang, X. Liu, and L. Zhao, "Recommending APIs based on API description and application context," in Proc. IEEE Int. Conf. Serv. Comput., 2018, pp. 331-334, doi: 10.1109/SCC.2018.00058.

[35] J. Li, X. Feng, Z. Li, X. Liu, H. Jiang, and Z. Xing, "Recommending APIs with context-aware knowledge from stack overflow posts," in Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng., 2020, pp. 1359-1371, doi: 10.1145/3377811.3380441.

[36] T. Nguyen, P. Vu, H. Pham, and T. Nguyen, "API recommendation using web search and stack overflow," in Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.: New Ideas and Emerging Results, 2019, pp. 25-28, doi: 10.1109/ICSE-NIER.2019.00015.

[37] S. Gao, C. Chen, Z. Xing, Y. Ma, W. Song, and S. Lin, "A neural model for method name generation from functional description," in Proc. IEEE 26th Int. Conf. Softw. Anal. Evol. Reeng., 2019, pp. 414-421, doi: 10.1109/SANER.2019.8668034.

[38] C. Chen, S. Gao, and Z. Xing, "Mining analogical libraries in Q&A discussions – incorporating relational and categorical knowledge into word embedding," in Proc. IEEE 23rd Int. Conf. Softw. Anal. Evol. Reeng., 2016, pp. 338-348, doi: 10.1109/SANER.2016.21.

[39] S. Gao, C. Chen, Z. Xing, Y. Ma, W. Song, and S. Lin, "A neural model for method name generation from functional description," in Proc. IEEE 26th Int. Conf. Softw. Anal. Evol. Reeng., 2019, pp. 414-421, doi: 10.1109/SANER.2019.8668034.

[40] G. Huang, X. Xia, T. Zhang, and Y. Zhou, "API method recommendation via explicit matching of functionality verb phrases," in Proc. ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas., 2019, pp. 1-10, doi: 10.1109/ESEM.2019.8870148.

[41] G. Huang, X. Xia, T. Zhang, and Y. Zhou, "API method recommendation via explicit matching of functionality verb phrases," in Proc. ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas., 2019, pp. 1-10, doi: 10.1109/ESEM.2019.8870148.

[42] X. Liu, L. Wang, C. Liu, and H. Xu, "Recommending API-mashup construction with integrating user interest and structural hole spanner," IEEE Access, vol. 6, pp. 66774-66786, 2018, doi: 10.1109/ACCESS.2018.2878708.

[43] L. Bao, X. Xia, D. Lo, and G. Murphy, "A large scale study of long-time contributor prediction for GitHub projects," IEEE Trans. Softw. Eng., vol. 47, no. 10, pp. 2225-2242, Oct. 2021, doi: 10.1109/TSE.2019.2918536.

[44] Z. Zhong, T. Wang, G. Yin, J. Yu, C. Yang, and H. Wei, "API recommendation for mashup development based on matrix factorization and association rule mining," in Proc. IEEE Int. Conf. Web Serv., 2017, pp. 708-715, doi: 10.1109/ICWS.2017.85.

[45] Z. Zhong, T. Wang, G. Yin, J. Yu, C. Yang, and H. Wei, "API recommendation for mashup development based on matrix factorization and association rule mining," in Proc. IEEE Int. Conf. Web Serv., 2017, pp. 708-715, doi: 10.1109/ICWS.2017.85.

[46] G. Huang, X. Xia, T. Zhang, and Y. Zhou, "API method recommendation via explicit matching of functionality verb phrases," in Proc. ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas., 2019, pp. 1-10, doi: 10.1109/ESEM.2019.8870148.

[47] L. Bao, X. Xia, D. Lo, and G. Murphy, "A large scale study of long-time contributor prediction for GitHub projects," IEEE Trans. Softw. Eng., vol. 47, no. 10, pp. 2225-2242, Oct. 2021, doi: 10.1109/TSE.2019.2918536.

[48] M. Rahman, C. Roy, and D. Lo, "RACK: Automatic API recommendation using crowdsourced knowledge," in Proc. IEEE 23rd Int. Conf. Softw. Anal. Evol. Reeng., 2016, pp. 349-359, doi: 10.1109/SANER.2016.65.

[49] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim, "Are code examples on an online Q&A forum reliable? A study of API misuse on stack overflow," in Proc. IEEE/ACM 40th Int. Conf. Softw. Eng., 2018, pp. 886-896, doi: 10.1145/3180155.3180260.

[50] B. Du, H. Tong, J. Bian, X. Bai, and T. Huang, "Personalized API recommendation via heterogeneous information network embedding," IEEE Access, vol. 7, pp. 101924-101935, 2019, doi: 10.1109/ACCESS.2019.2930725.

[51] X. Niu, Z. Zheng, J. Chen, X. Jia, and J. Wu, "API recommendation for mashup development based on API usage patterns and domain knowledge," J. Syst. Softw., vol. 156, pp. 35-47, Oct. 2019, doi: 10.1016/j.jss.2019.06.079.

[52] W. Xu, X. Liu, and Y. Gong, "Recommending API functions to developers using collaborative filtering," in Proc. 29th ACM/IEEE Int. Conf. Autom. Softw. Eng., 2014, pp. 331-342, doi: 10.1145/2642937.2642988.

[53] G. Huang, X. Xia, T. Zhang, and Y. Zhou, "API method recommendation via explicit matching of functionality verb phrases," in Proc. ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas., 2019, pp. 1-10, doi: 10.1109/ESEM.2019.8870148.

[54] G. Huang, X. Xia, T. Zhang, and Y. Zhou, "API method recommendation via explicit matching of functionality verb phrases," in Proc. ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas., 2019, pp. 1-10, doi: 10.1109/ESEM.2019.8870148.

[55] B. Du, H. Tong, J. Bian, X. Bai, and T. Huang, "Personalized API recommendation via heterogeneous information network embedding," IEEE Access, vol. 7, pp. 101924-101935, 2019, doi: 10.1109/ACCESS.2019.2930725.

[56] T. Nguyen, P. Vu, H. Pham, and T. Nguyen, "API recommendation using web search and stack overflow," in Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.: New Ideas and Emerging Results, 2019, pp. 25-28, doi: 10.1109/ICSE-NIER.2019.00015.

[57] S. Gao, C. Chen, Z. Xing, Y. Ma, W. Song, and S. Lin, "A neural model for method name generation from functional description," in Proc. IEEE 26th Int. Conf. Softw. Anal. Evol. Reeng., 2019, pp. 414-421, doi: 10.1109/SANER.2019.8668034.

[58] Y. Hu, Q. Shen, D. Jiang, J. Wang, Q. Zhu, and C. Xu, "APIRec: An approach to recommend APIs for mashup development," J. Syst. Softw., vol. 170, p. 110774, Dec. 2020, doi: 10.1016/j.jss.2020.110774.

[59] X. Xia, L. Bao, D. Lo, P. Kochhar, A. Hassan, and Z. Xing, "What do developers search for on the web?" Empir. Softw. Eng., vol. 22, no. 6, pp. 3149-3185, Dec. 2017, doi: 10.1007/s10664-017-9514-4.

[60] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," Communications of the ACM, vol. 63, no. 12, pp. 54-63, 2020, doi: 10.1145/3381831.

[61] T. Nguyen, P. Vu, H. Pham, and T. Nguyen, "API recommendation using web search and stack overflow," in Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.: New Ideas and Emerging Results, 2019, pp. 25-28, doi: 10.1109/ICSE-NIER.2019.00015.

[62] J. Li, X. Feng, Z. Li, X. Liu, H. Jiang, and Z. Xing, "Recommending APIs with context-aware knowledge from stack overflow posts," in Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng., 2020, pp. 1359-1371, doi: 10.1145/3377811.3380441.

[63] T. Nguyen, P. Vu, H. Pham, and T. Nguyen, "API recommendation using web search and stack overflow," in Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.: New Ideas and Emerging Results, 2019, pp. 25-28, doi: 10.1109/ICSE-NIER.2019.00015.

[64] M. Aghajani, G. Gousios, N. Niu, T. Menzies, and M. Nagappan, "An empirical study on API recommendation: Challenges and opportunities," IEEE Trans. Softw. Eng., pp. 1-1, 2021, doi: 10.1109/TSE.2021.3087991.

[65] J. Li, X. Feng, Z. Li, X. Liu, H. Jiang, and Z. Xing, "Recommending APIs with context-aware knowledge from stack overflow posts," in Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng., 2020, pp. 1359-1371, doi: 10.1145/3377811.3380441.

[66] X. Liu, L. Wang, C. Liu, and H. Xu, "Recommending API-mashup construction with integrating user interest and structural hole spanner," IEEE Access, vol. 6, pp. 66774-66786, 2018, doi: 10.1109/ACCESS.2018.2878708.

[67] T. Nguyen, P. Vu, H. Pham, and T. Nguyen, "API recommendation using web search and stack overflow," in Proc. IEEE/ACM 41st Int. Conf. Softw. Eng.: New Ideas and Emerging Results, 2019, pp. 25-28, doi: 10.1109/ICSE-NIER.2019.00015.

[68] J. Li, X. Feng, Z. Li, X. Liu, H. Jiang, and Z. Xing, "Recommending APIs with context-aware knowledge from stack overflow posts," in Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng., 2020, pp. 1359-1371, doi: 10.1145/3377811.3380441.

[69] X. Niu, Z. Zheng, J. Chen, X. Jia, and J. Wu, "API recommendation for mashup development based on API usage patterns and domain knowledge," J. Syst. Softw., vol. 156, pp. 35-47, Oct. 2019, doi: 10.1016/j.jss.2019.06.079.

[70] G. Huang, X. Xia, T. Zhang, and Y. Zhou, "API method recommendation via explicit matching of functionality verb phrases," in Proc. ACM/IEEE Int. Symp. Empir. Softw. Eng. Meas., 2019, pp. 1-10, doi: 10.1109/ESEM.2019.8870148.

[71] B. Du, H. Tong, J. Bian, X. Bai, and T. Huang, "Personalized API recommendation via heterogeneous information network embedding," IEEE Access, vol. 7, pp. 101924-101935, 2019, doi: 10.1109/ACCESS.2019.2930725.

[72] Y. Hu, Q. Shen, D. Jiang, J. Wang, Q. Zhu, and C. Xu, "APIRec: An approach to recommend APIs for mashup development," J. Syst. Softw., vol. 170, p. 110774, Dec. 2020, doi: 10.1016/j.jss.2020.110774.