OS^3: The Art and the Practice of Searching for Open-Source Serverless Functions

Sarvesh Bhatnagar, Zhengquan Li, Zheng Song
Dept. of Computer and Information Science
University of Michigan at Dearborn
Dearborn, Michigan, USA
{sarveshb, zqli, zhesong}@umich.edu

Eli Tilevich
Dept. of Computer Science
Virginia Tech
Blacksburg, Virginia, USA
tilevich@cs.vt.edu

***Abstract*—Serverless computing enables service developers to focus on creating useful services, without being concerned about how these services would be deployed and provisioned. Many developers reuse existing open-source serverless functions to create their own functions. However, existing technologies for searching open-source software repositories have not taken into consideration the unique features of serverless functions. This paper presents a novel approach to searching for serverless functions, called Open-Source Serverless Search (OS^3) that maximizes the utility of the returned serverless functions by (1) basing the search process on both descriptive keywords and library usages, thus increasing the search results' precision and completeness; (2) filtering and ranking the search results based on the software license, to accommodate the unique requirements of deploying serverless functions on dissimilar platforms, including cloud and edge computing. Implemented in 3K lines of Python, with a search space of 5,981 serverless repositories from four major serverless platforms, OS^3 outperforms existing search approaches in terms of the suitability of the search results, based on our evaluation with realistic use cases.**

***Index Terms*—Serverless Computing, Open Source Code Search, Serverless Dataset**

I. Introduction

Serverless computing provides powerful abstractions of the underlying infrastructure management (e.g., load balancing, scaling-on-demand, etc.), rapid prototyping, and flexible pay-as-you-go model, thus relieving application developers from the concerns related to the low-level aspects of deploying and provisioning remote service code. Due to seemingly infinite cloud resources allocated at runtime, this model allows for the infinite elasticity of serverless functions, so developers can focus on the high-level design aspects of service-oriented applications [1]. From the software engineering standpoint, serverless promotes reusability, as service developers naturally produce modular service components, to be reused across multiple projects, rather than monolithic services, tailored for specific projects.

To be able to compose their software solutions out of existing serverless functions, application developers need to be able to find serverless functions that are *the most suitable for the task at hand*. As it turns out, this is a formidable task due to two main complications. First, the high heterogeneity of vendors and platforms rules out a one-stop platform for service developers to share their serverless functions. The serverless ecosystem is highly fragmented across multiple infrastructure providers, including Microsoft Azure, IBM Cloud, AWS, Google Cloud, and open-source serverless frameworks [1] that support multiple infrastructure providers. AWS [2] maintains its own platform for developers to share open source repositories, Google Cloud maintains 51 sample functions [3], while developers of other serverless platforms open source their functions on Github. Second, the existing search facilities for serverless functions rely on keyword-based searching heuristics, which alone are insufficient to find the functions based on their suitability for a given task.

To be able to effectively search for serverless functions, a search facility must take into account their unique characteristics such as (1) library usage, a critical issue for serverless as compared with traditional services; and (2) software licensing, which imposes legal restrictions on how serverless functions can be used. Without considering these characteristics, existing search facilities cannot identify the most fitting serverless functions. Driven by these insights, this paper presents a novel approach to searching for serverless functions. Dubbed OS^3, our approach maximizes the suitability of returned serverless functions by incorporating the unique characteristics above. For (1), it improves the precision and completeness of the search results by complementing keyword-based search results with library usage; for (2), it filters out license-problematic repositories based on expected usage scenarios. The reference implementation of OS^3 searches for open-source serverless functions across multiple vendor platforms. Its search space comprises 5,981 serverless repositories, filtered out from over 60K repositories, collected for 4 major vendors from GitHub. OS^3 not only improves the precision and completeness of keyword-based search, but also identifies repositories protected by appropriate software licenses, thus returning search results that are most suitable for the task at hand. This paper makes the following contributions:

1) We collect a dataset of 60K serverless repositories for

[1] http://github.com/serverless/serverless/
[2] http://serverless- repo.aws.amazon.com/applications
[3] http://cloud.google.com/functions/docs/samples/

4 major serverless vendors from GitHub[4]. We carefully study the data and identify several unique aspects of searching for serverless functions.

2) We carefully analyze the key features of the collected dataset, including the heretofore unexplored angle of software licensing. Our findings suggest that the library usages in serverless repositories correlate closely with their functionalities; further, certain licenses might restrict how serverless repositories can be used in edge computing environments.
3) We build the first search engine for cross-platform, open-source serverless functions. We introduce a clustering-based algorithm that improves keyword-based search by considering library usage; it refines the search results of existing search algorithms by removing false positives and appending repositories that were originally missing. It also filters out repositories whose licenses make them inapplicable for edge/cloud computing environments.

The rest of the paper is organized as follows: Section II introduces our data collection and processing methodology as well as discusses how a search facility for serverless functions should take into account their unique characteristics. Section III presents the design and implementation details of our search engine, respectively. Section IV compares our engine's search results with those of existing search engines. Section V discusses related works. Section VI concludes the paper and discusses future work directions.

II. Data Collection and Analysis

In this section, we discuss the main considerations we made when collecting our dataset. Serverless repositories have not been studied in great detail, so the collected datasets of serverless functions are not sufficiently comprehensive [2], [3]. To fill this knowledge gap, we collected a dataset of over 60K serverless repositories from GitHub and analyzed the resulting dataset to inform the design of our search engine.

A. Data Collection and Cleaning

Having examined numerous datasets, we identified the recently published Wonderless dataset [3] with 1,877 serverless repositories to be the most comprehensive among its peers. Despite its large size, this dataset's only source of repositories is through the Serverless framework and the criteria for cleaning data are overly strict. With these criteria in place, Wonderless removes multiple repositories that might still have a recommendation value and as such be useful to developers for reuse considerations.

Hence, to create our dataset, we obtained data through different means rather than only via the Serverless framework as is the case with the Wonderless Dataset. We consider different kinds of configuration files and their peculiarities to search for repositories on GitHub for different platforms. To identify the repositories that use the Serverless framework, we search for a configuration file "`serverless.yml`"; the ones that use AWS, "`template.yml`"; the ones that use Azure, "`function.json`"; and the ones that use IBM functions "`manifest.yml`". Some configuration files (e.g., `function.json`, `manifest.yml`) are frequently seen in software repositories that are not related to serverless. To avoid collecting irrelevant repositories, we use configuration peculiarities that help in validating whether a repository is actually used for implementing a serverless function. We compiled these peculiarities based on the documentation for their respective platforms (Table I). We also remove toy repositories that include keyword phrases such as `example`, `demo`, and `test` in their Readme files. By following this process, we collected a total of 67,744 repositories with the following composition: 29,995 for the Serverless framework, 14,164 for AWS, 21,523 for Azure and 2,062 for IBM.

TABLE I
Platform Validation using Config files

Platform	Configuration File	Validation Check
AWS	template.yml	AWSTemplateFormatVersion
Azure	function.json	direction
IBM	manifest.yml	actions

We then filter all repositories by first removing the unlicensed repositories, a step not performed when creating the Wonderless dataset. The removal of unlicensed repositories is important for recommending serverless functions, as it is the owner alone that reserves all rights for an unlicensed repository, which as such cannot be reused or modified without the owner's explicit permission. Furthermore, we remove all duplicate repositories.

Having performed all the aforementioned cleaning and filtering steps, we narrow down our dataset to a total of 5,981 repositories with 5,220 repositories using the Serverless framework, 498 developed for AWS, 241 for Azure and 22 for IBM. We note that our numbers of repositories for different platforms are in alignment with the popularity of these platforms [4].

B. Usages of Externally Managed API

We observe that 88% of serverless functions depend on externally managed API's, including S3 [5], DynamoDB [6], Lex [7], Polly [8], SQS [9] etc. Developers use DynamoDB for persist state updates, S3 for storage, and Lex for chatbots.

We also observe that developers of serverless functions are more likely to avoid importing unnecessary libraries as compared with other software developers. On average for serverless functions, we find 2.8 library imports per serverless function, whereas for standard software, the number is much higher with an average of 11.6 library imports per project [5]. This difference is mainly due to the following reasons: 1)

[4]https://github.com/edgeumd/serverless_dataset

[5]https://aws.amazon.com/s3/
[6]https://aws.amazon.com/dynamodb/
[7]https://aws.amazon.com/lex/
[8]https://aws.amazon.com/polly/
[9]https://aws.amazon.com/sqs/

serverless functions are usually developed following the microservice software architecture, in which monolithic software is partitioned into smaller components, so each component is encapsulated as a serverless function. 2) the pay-as-you-go pricing model of serverless requires that serverless developers be very cautious about the size of their serverless containers, as a larger container takes longer to ship and more memory to deploy, thus increasing its execution costs. These reasons force developers to make their code efficient and only include those libraries that are truly used in the functionality of a serverless function.

Hence, one feature of serverless functions is that externally managed libraries are widely used and such library usages are more correlated with their functionalities as compared with other types of software. This observation motivates us to consider serverless repositories' library usages in OS^3.

Most of the aforementioned externally managed APIs have their corresponding versions on other platforms as well. We have compiled representative sets of equivalent APIs for different platforms in Table II.

TABLE II
EQUIVALENT MANAGED SERVICES

	AWS	**Azure**	**IBM**
1	S3	Blob Storage	COS
2	DynamoDB	Cosmos DB	DB2
3	SNS	Event Grid	Cloud Event Notification Service
4	SQS	Storage Queues	MQ
5	SES	SendGrid	APP Connect
6	Kinesis	Event Hubs	Streams
7	Lex	Bot Service	Watson
8	Polly	Text to Speech	Watson Text to Speech

These equivalent APIs can help finding similar serverless functions developed for different platforms. For example, if one serverless function uses `S3`, `SNS`, and `Polly` and another function uses `COS`, `Cloud Event notification Service`, and `Watson Text to Speech`, it is very likely that these two functions are developed for AWS and IBM respectively, and they provide similar functionalities. This observation motivates us to consider equivalent APIs in OS^3.

C. License considerations for Edge and Cloud Computing

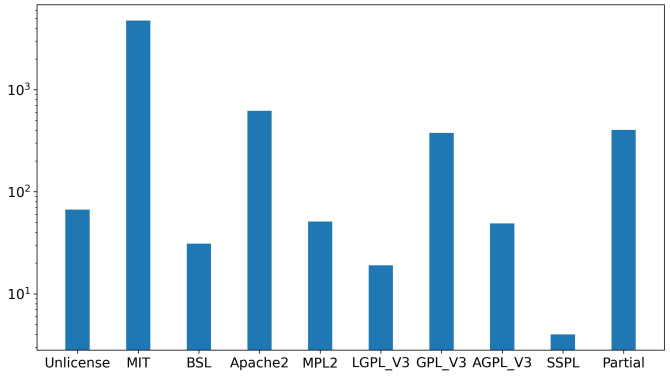

Fig. 1. License Distribution In Serverless Functions (log scale)

In our collected repositories the license distribution can be seen in Fig. 1, where the licenses are arranged in the order of restriction, with the left-most licenses being the least restrictive. The "Unlicense" is actually a license that poses the least restrictions on users. Although a large part of serverless functions uses one of the least restrictive licenses, most are either unlicensed or are toy repositories. To top that, of the remaining repositories we found that about 9.5% of the repositories use licenses toward the restrictive end (from MPL2 to SSPL) whereas about 7.7% are only partially licensed. Partially licensed repositories have some components that can be reused while some are not usable. This is because many popular software licenses grant permissions for developers to use and modify open-source software under the terms of certain agreements. For example, some licenses (Apache2, MIT) allow for commercial use of open source software along with any software built upon it. Some license (LGPL) requires an application that is built upon the software they protect to open source its code as well if the application is distributed to end users. Some licenses (LGPL, AGPL) require an application to open source its entire code if any part of the application is built upon open source software protected by the licenses. When it comes to reusing serverless repositories, the software license considerations share some similarities but also have noticeable differences with cloud services. Different from traditional software, cloud-based services require special considerations for software licenses for two reasons: 1) as services are deployed in the cloud, they may not be considered as "being distributed to the end users." Therefore, a cloud service can choose not to open source its code even if it is developed upon open source software protected by a restrictive license; 2) if a monolithic service is partitioned into microservices, developers of a cloud service only need to open source the part of the code, or more precisely, the microservice that uses the open source repository protected by the restrictive license, rather than open sourcing all microservices. Similar to cloud services, using a serverless function can avoid the need to open source the entire application while deploying a serverless function protected by a restrictive license in the cloud does not require to open source its code. However, serverless functions may be deployed on edge servers, owned by individuals, so the deployment becomes subject to the "being distributed to the end users" clause. Hence, we argue that a search engine for serverless functions should consider the planned application deployment scenarios. If a developer has no plans to open source their modifications on a serverless repository with a license starting from MPL2 to Partial, the resulting code cannot be deployed in edge environments.

III. OS^3 DESIGN AND IMPLEMENTATION

We base the design of our OS^3 system upon the general traits of serverless functions and key insights we derived from analyzing the collected data. In particular, OS^3 is intended to be able to leverage the unique features of serverless functions in order to (1) eliminate incorrect results, (2) uncover the

ignored results that are relevant to the search query, and (3) give higher rankings to high-usability serverless functions.

A. Incorporating Existing Search Algorithms

Developers currently apply multiple keyword based mature techniques to search for and recommend open source functions, including keyword matching [6], topic modeling-based search [7], and Word2Vec-based search [8]. The keyword matching search generally matches keywords within Readme files or their descriptions.

Although existing serverless repositories, including Github and AWS Serverless Application Repository (SAR), apply these techniques to search for and recommend serverless functions, these techniques tend to overgeneralize their search results, thus causing high false positive rates [9].

OS^3 aims to keep the underlying search algorithm as is while improvising the results to find valid and more accurate results for serverless functions. The reference implementation provides two keyword based search algorithms: keyword-matching and word-vector based. For keyword-matching, we first preprocess the Readme files by removing stopwords, performing stemming, and then we match the stem of keywords with Readme and code files. To determine the similarity, we calculate the raw scores with a preference for the highest match. Similarly, we also implement a search algorithm that is based on the Word2Vec [8] algorithm and compares word vectors for the user's search query and the corresponding Readme files of serverless repositories.

B. Improving Search by Exploiting Serverless Peculiarities

OS^3 takes advantage of serverless-specific peculiarities to improve the search results. The desirable properties of serverless functions that led to their unprecedented growth (i.e., infinite perceived elasticity, reduced DevOps, and the pay-as-you-go model) also impose certain restrictions on the developer. As revealed in our study, the usages of externally managed libraries are more correlated with the functionality of a serverless function. We use this peculiarity of serverless functions to find more comprehensive results and filter out inaccurate results.

In particular, we take as input the search results of an existing search and a ranking algorithm, and then refine the given search results by following these steps:

1) Cluster the repositories in the search results by their library usages.
2) Find the cluster $\mathcal{C}$ with the highest energy. We define the energy of a cluster of serverless functions $f \in \mathcal{C}$ as $e = \frac{\sum_{f \in \mathcal{C}} R_f}{|\mathcal{C}|}$, where R_f denotes the ranking of function f in the search result, with a higher R_f indicating the function is more recommended.
3) Extract the most identical libraries from the selected cluster and expand the equivalent libraries. By extracting the libraries from the selected cluster, we identify what libraries are used in the cluster of serverless functions that are most likely to fit the user's search intent. We then expand the libraries with equivalent libraries from other vendors (see Table II).
4) Filter out the serverless functions that use identical libraries insufficiently.
5) Append the serverless functions excluded in the original result if they contain libraries in the most identical set.
6) Rerank based on useage with the help of licenses.

C. Filtering and Reranking

Our approach to filtering and reranking repositories is guided by their intended usage scenarios. We rely on the attached software licenses to identify useful repositories and filter out the rest. We observe that licenses can play an important role when selecting a repository. Our analysis identifies three main use cases: a repository for edge distribution, a repository for cloud-based services, and a repository for open-source development. For edge distribution, no repository can have licenses from MPL2 to Partial. This restriction is due to edge distribution requiring developers to open-source their complete code. Similarly, when cloud distribution is considered and the recommended repository is being provided to the user as a service only, we filter out repositories from SSPL to Partial. Finally, in the default case of open-source development, we rerank the repositories, so they are recommended based on how restrictive their license is, with the less restrictive licenses placed first.

D. Case Study

Our reference implementation of OS^3 comprises about 3,000 lines of Python code. Fig. 2 describes how OS^3 works internally with real-world data. The search query in question was "`machine analysis and learning`" for which we received 129 search results using keyword-based search. We further applied the Louvain algorithm to detect underlying clusters in the results. Note that there were 8 clusters detected and each cluster had more repositories than depicted in the figure. We apply Step 2 in the search result refining algorithm to detect clusters with the highest energy (cluster 2). Using the detected cluster we find the libraries with the most count (*numpy*, *pandas*). We also find *SQS* being used which we expand with *MQ* and *Azure Storage Queue*.

After finding the libraries, we use them to filter out the search results where we remove *Pollaris* and *LambdaMailer*, as they have no common libraries. Furthermore, we also use the common libraries to discover new repositories from our global repository database. We were able to detect 3 repositories (*data-analysis-aws*, *smartguard*, *mdm-night-owl*) of which each had *pandas* and *numpy*. We then combine these filtered repositories and new recommendations together while keeping the existing recommendation intact and filling the filtered repositories (*Pollaris*, *LambdaMailer*). In the cases in which not enough repositories were found to fill in the missing places, we take the preceding recommendations in the original search result. Finally, we exemplify how the recommendations might change based on different use cases. e.g. usage for edge based on discussions on licenses. We filtered out *Hackathon*

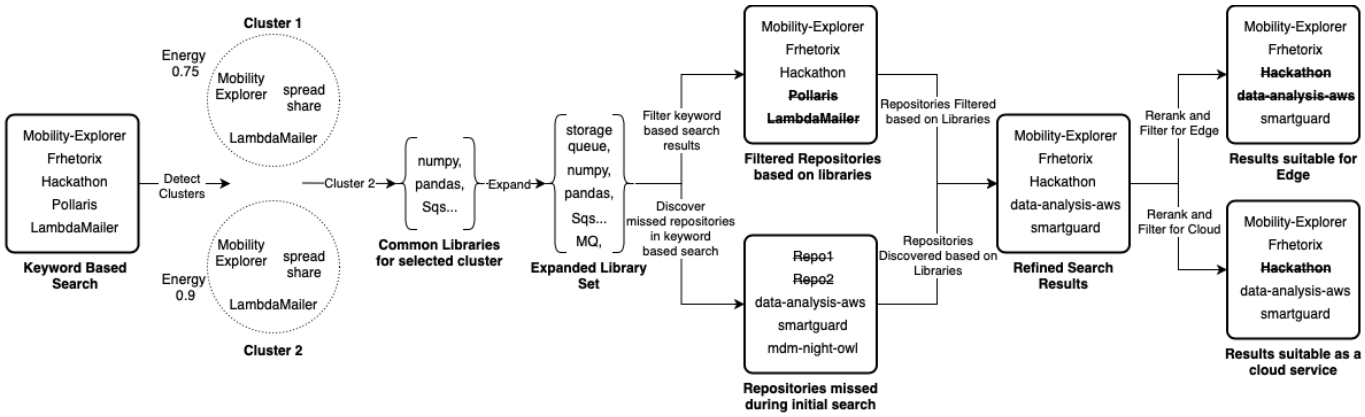

Fig. 2. Case Study: Searching for "Machine Analysis and Learning" Serverless Functions

and *data-analysis-aws* due to the overly restrictive nature of their licenses.

IV. Evaluation

To evaluate the effectiveness of OS^3, we seek answers to the following research questions:

Research Question 1: Compared with basic keyword-based search approaches, how much can OS^3 improve precision?

Research Question 2: Can OS^3 accurately discover new repositories that are missing by the basic keyword-based search approaches?

Research Question 3: Can license-based filtering improve the suitability of the search results?

A. Experimental Design

To create a list of search queries that we will use to evaluate the performance of **keyword-based search** against **keyword-based + OS^3**, we go through multiple research [10]–[13] collecting a total of 18 queries. These queries are formulated based on either using popular repositories for serverless or actively researched repositories. We run the underlying search algorithm alone and with OS^3 against these queries to find recommendations. Out of those recommendations, we select the top 5 results, disregarding the rest of them.

These results are then manually checked against their searched queries to determine the number of false positives in the top five recommendations. Two reviewers manually examined each search result. After carefully reading the description file and the source code of the repository, each reviewer individually decided whether the repository fit the search intent. If the two reviewers held different opinions toward one repository, a third reviewer was asked to help make the final decision.

Furthermore, we also check how many new repositories added by OS^3 fit the search query, as well as how many repositories in the final recommendation results are filtered out due to inappropriate licenses.

B. Experimental Results and Discussion

Fig. 3 and Fig. 4 show the performance of OS^3 with two keyword-based search algorithms, the Keyword Matching and the Word vector-based search, respectively. In both cases, we

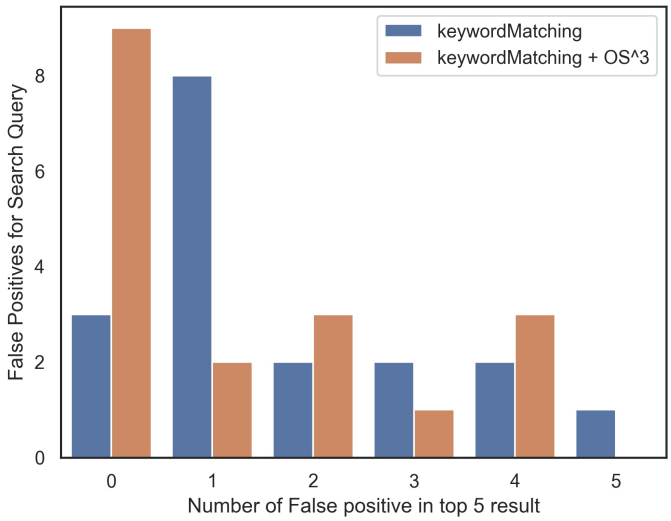

Fig. 3. False Positives For Keyword Matching with and without OS^3

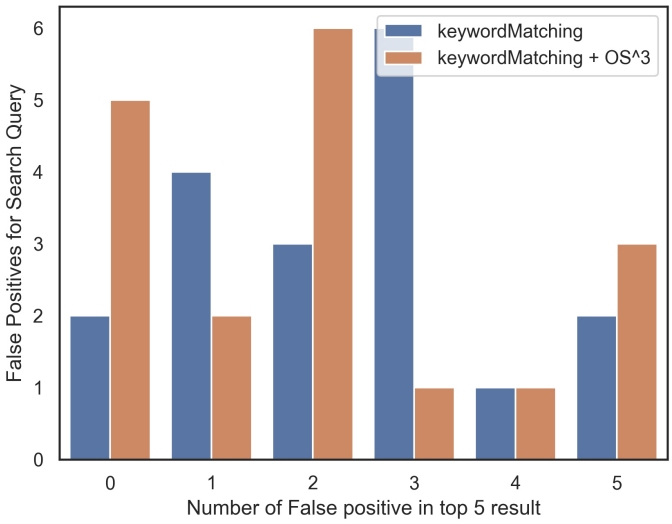

Fig. 4. False Positives For Word Vector Based Search with and without OS^3

can see the increased right skew in the graph. This right skew in the graph signifies the reduction in false positives during recommendation. Our results indicate that using OS^3 increases the precision of Keyword matching by 8.89% (from 65.55%

to 74.44%) and the precision of Word Vector-based search by 8.23% (from 55.29% to 63.52%.)

We also manually inspected the new repositories recommended by OS^3 and found that 70.58% of them fit the search keywords. Nevertheless, sometimes when the original research results are not accurate, the new repositories recommended by OS^3 are inaccurate. For example, the original results for searching "`sentiment analysis`" have a high false positive rate, so the new repositories discovered by OS^3 are inaccurate as well, including *serverless-cognito*, *python-lambda-monorepo* and *aws-twitter-translate-bot*.

The analysis of the licenses used by the recommended repositories revealed that about 11% of the initial search results are unusable. In particular, about 10 repositories in 90 results for the 18 queries are unusable for edge environments. Having the top 5 results of a search query contain 11% of unusable repositories is rather high, particularly in light of only 17.2% of repositories in total belonging to the MPL2 to Partial categories (Fig. 1).

We answer the research questions above as follows:
RQ1: Using OS^3 over the underlying search algorithms does increase the precision by a decent amount; in our evaluations, we find that to be by about 8.89% for the keyword matching algorithm and 8.23% for the word vector-based approach.
RQ2: We find that OS^3 is able to discover new repositories not found by the underlying search with an accuracy of 70.58%.
RQ3: In our evaluation, we find that about 11% repositories recommended can be unusable when using them for real-life applications among the highly ranked repositories. Furthermore, the sample set for hard-to-use licenses is about 17.2%, thereby giving further validity to the importance of considering software licensing when recommending repositories.

V. Related Work

There have been many approaches for code-based search and recommendation that take into account the semantics or structure of code snippets. One such recent attempt is Aroma [14], which focuses on finding contextual code samples dissimilar from each other within the structural similarity scope, and thereby helps in finding different ways a piece of code has been written by different developers. Another approach is to use code-to-code pattern matching that could suggest similar code using code snippets as a query. Researchers have also tried using AST trees to enable code vector embedding that can be used for code recommendation [15]. However, such semantics-based or structure-based approaches are language dependent and require dedicated parsers for different languages. Compared to the code-based search approach, our underlying keyword-based search is naturally more suitable for language-agnostic serverless functions. Besides, we also consider the license requirements for using serverless functions in edge computing environments [16], [17].

VI. Conclusion

To address the emerging need for searching and recommending serverless functions, this paper introduces OS^3, a novel search engine custom-tailored for serverless functions. Our engine is optimized for searching serverless repositories by focusing on the following aspects: 1) it bases the search process on both descriptive keywords and library usages, thus increasing the precision and completeness of the search results; 2) it filters and ranks the search results on the software license and where the serverless function will be executed. We have concretely implemented OS^3 and evaluated it on realistic use cases. Our evaluation indicates that our search engine can become a useful tool in developing serverless applications.

Acknowledgments

This research is supported by NSF Grants #2104337 and #2232565.

References

[1] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: A survey of opportunities, challenges, and applications," *ACM Computing Surveys (CSUR)*, 2019.

[2] S. Eismann, J. Scheuner, E. Van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. Abad, and A. Iosup, "The state of serverless applications: Collection, characterization, and community consensus," *IEEE Transactions on Software Engineering*, 2021.

[3] N. Eskandani and G. Salvaneschi, "The wonderless dataset for serverless computing," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 565–569.

[4] I. Pavlov, S. Ali, and T. Mahmud, "Serverless development trends in open source: a mixed-research study," 2019.

[5] "60678 libraries on github," www.overops.com/blog/we-analyzed-60678-libraries-on-github-here-are-the-top-100/, accessed: 2022-5-10.

[6] P. Chaudhari and M. L. Das, "Keysea: Keyword-based search with receiver anonymity in attribute-based searchable encryption," *IEEE Transactions on Services Computing*, 2020.

[7] J. Zhao, J. X. Huang, H. Deng, Y. Chang, and L. Xia, "Are topics interesting or not? an lda-based topic-graph probabilistic model for web search personalization," *ACM Trans. Inf. Syst.*, vol. 40, no. 3, dec 2022. [Online]. Available: https://doi.org/10.1145/3476106

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv:1301.3781*, 2013.

[9] T. Montecchi, D. Russo, and Y. Liu, "Searching in cooperative patent classification: Comparison between keyword and concept-based search," *Advanced Engineering Informatics*, vol. 27, no. 3, pp. 335–345, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474034613000219

[10] A. Hall and U. Ramachandran, "An execution model for serverless functions at the edge," in *Proceedings of the International Conference on Internet of Things Design and Implementation*, 2019, pp. 225–236.

[11] B. Carver, J. Zhang, A. Wang, and Y. Cheng, "In search of a fast and efficient serverless dag engine," in *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*. IEEE, 2019, pp. 1–10.

[12] J. Spillner, "Quantitative analysis of cloud function evolution in the aws serverless application repository," *arXiv:1905.04800*, 2019.

[13] M. Obetz, S. Patterson, and A. Milanova, "Static call graph construction in AWS lambda serverless applications," in *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

[14] S. Luan, D. Yang, C. Barnaby, K. Sen, and S. Chandra, "Aroma: Code recommendation via structural code search," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–28, 2019.

[15] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, 2019.

[16] M. Le, Z. Song, Y.-W. Kwon, and E. Tilevich, "Reliable and efficient mobile edge computing in highly dynamic and volatile environments," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 2017, pp. 113–120.

[17] Z. Song and E. Tilevich, "A programming model for reliable and efficient edge-based execution under resource variability," in *2019 IEEE International Conference on Edge Computing (EDGE)*. IEEE, 2019, pp. 64–71.