Investigating Technical Debt Types, Issues, and Solutions in Serverless Computing

Hasini Sumalee Perera
University of Saskatchewan
Saskatoon, Canada
bhn169@mail.usask.ca

Zadia Codabux
University of Saskatchewan
Saskatoon, Canada
zadiacodabux@ieee.org

Fabio Palomba
University of Salerno
Fisciano, Italy
fpalomba@unisa.it

***Abstract*—Serverless computing is a cloud execution model where developers run code and the server management is handled by the cloud provider. Serverless computing is increasingly gaining popularity as more systems adopt it to enhance scalability and reduce operational costs. While it has numerous benefits, it also embodies unique challenges inherent to serverless computing. One such challenge is Technical Debt (TD), which is exacerbated by the complexities of the serverless paradigm. While prior work has investigated the activities and bad practices that lead to TD in serverless computing, there remains a gap in understanding how TD manifests, the challenges it poses, and the solutions proposed to address TD issues in serverless systems. This study aims to investigate TD in the serverless context using Stack Overflow (SO) as a knowledge base. We collected 78,867 serverless questions on SO and labeled them as TD or non-TD using deep learning. We further conducted a deeper exploration to identify types of TD in serverless settings, issues and proposed solutions, and also explored TD in the code snippets. We found that 37% of serverless questions on SO are TD-related, and that the majority of code snippets contained code smells and security vulnerabilities. We also identified six serverless-specific issues. Our research highlights the need for tools that can effectively detect TD in serverless applications.**

***Index Terms*—Technical Debt, Serverless Computing, Crowdsourcing, Stack Overflow, Knowledge Repository**

I. Introduction

Serverless computing is a cloud service model in which the service provider controls the provisioning and management of servers. This architecture allows code to be executed without direct control over underlying resources [1]. The most prevalent type of serverless computing is Function-as-a-Service (FaaS), which uses platforms such as AWS Lambda, Azure Functions, and Google Cloud Functions. As software architecture patterns have evolved from monolithic architectures to microservices, and then to serverless functions, organizations seek improved maintainability and adaptability for their systems [2]. Even though adopting serverless computing offers benefits in scalability and efficiency, it can magnify long-term challenges in system quality and performance.

Considering the similarities between serverless computing and microservices, it is reasonable to anticipate that maintainability issues in serverless computing will be significant and ultimately lead to Technical Debt (TD), as is the case with microservices [3]. TD arises from the compromises developers make to expedite development. TD may not be an issue right away, but can degrade the system's quality and performance over time, if not managed [4]. In serverless computing, TD can accrue due to rushed architectural decisions, inadequate knowledge of services, minimal process automation, and neglected security practices [3], [2]. When developers encounter issues related to TD when implementing software, they frequently turn to community-driven technical Q&A forums like Stack Overflow (SO) to seek advice, share experiences, and discover solutions. SO hosts 23 million questions, 34 million answers, and 19 million users, and is used by 30% of software practitioners every day, making it an invaluable knowledge resource for software engineering researchers [5]. These publicly available discussions provide a rich source of data for understanding how TD manifests in real-world serverless applications and how practitioners attempt to resolve it.

Several studies examine the concept of serverless computing, the difficulties developers encounter during the implementation of serverless applications, and the bad practices that contribute to TD in serverless computing [6], [7], [2], [8]. Although existing research on serverless computing provides valuable insights, it primarily focuses on the benefits, drawbacks, and challenges of implementing serverless applications. As far as we are aware, there has been only one study [3] that focuses on TD in serverless computing, which was conducted by interviewing three experts and is limited to code, testing, and architectural debt. There is a lack of empirical evidence on how different TD types appear in real-world serverless applications and how they are addressed. Our research aims to bridge this gap by providing a comprehensive analysis of the types, subtypes, and solutions to TD questions in serverless computing, by using SO as a knowledge base. This can help provide guidelines to developers on how to minimize TD during the implementation of serverless applications. Moreover, it can help design more effective tools or strategies on how TD issues can be detected and managed in these rapidly evolving environments.

This study systematically analyzes discussions on SO about TD in serverless computing. We quantify the *prevalence* of TD and *categorize* its types within serverless architectures. Furthermore, we *identify TD issues* exclusive to serverless computing and evaluate the *proposed solutions* discussed by developers.

The contributions of our study are as follows:

1) The most common TD types in the serverless domain.
2) A catalog of TD types in the serverless domain.
3) A catalog of solution types for serverless TD questions.
4) An analysis of TD in code snippets of serverless questions to identify code smells and security vulnerabilities.
5) A comprehensive replication package[1].

II. Related Work

Investigating TD using SO. Kozanidis et al. [9] employed both automated and manual approaches to identify the questions related to TD. Questions extracted from SO were carefully evaluated to perform quantitative and qualitative analyses across various aspects, including TD types, question length, urgency, sentiment, and themes. They used machine learning to classify SO questions as TD or non-TD. Their research explored the possibility of using machine learning to identify TD and its types within SO questions. While their study investigates TD in general, we focus on identifying TD in serverless-related questions on SO. Edbert et al. [10] collected data from SO to explore the TD in security-related questions on SO. They extracted data from predefined security tags in Yang et al. [11], classified the extracted security questions as TD or non-TD, and explored the characteristics of security-related TD questions on SO. We followed a similar approach to their study, but we focused on investigating the issues and proposed solutions of serverless-related TD questions on SO. Titan et al. [12] used 14 terms related to architecture smells to extract relevant questions on SO, resulting in the collection of 207 questions. They utilized the grounded theory method to analyze these questions, focusing on different aspects of architectural smells. This paper examines discussions obtained through specific terms related to architectural smells.

Studies on Serverless Computing. Taibi et al. [2] interviewed experienced software engineers proficient in serverless technology to identify bad practices in serverless and propose solutions to address them. This process identified six bad practices related to applications developed using serverless architecture. Leitner et al. [7] conducted a mixed-method study to investigate serverless technologies in industry. Their findings included prevalent design patterns in serverless, as well as the perceived advantages and difficulties developers encounter when utilizing serverless computing. Wen et al. [8] conducted empirical research in which they manually evaluated 619 examples and analyzed 22,731 SO questions related to serverless computing to develop a taxonomy of serverless challenges. Baldini et al. [6] presented a programming paradigm along with additional relevant features of serverless computing. Additionally, they identified the types of application workloads that are suitable for serverless computing platforms.

TD in Serverless. Exploring TD in serverless applications is a relatively understudied topic. Lenarduzzi et al. [3] interviewed three experts to determine the activities that contribute to the accumulation of TD across code, test, and architecture debt. They investigated how TD is accrued in serverless functions, identifying a preliminary set of issues that are indicators of TD. Their analysis shed light on the types of TDs most likely to affect serverless computing.

Studies on Microservices. Contrary to serverless computing, Microservices, the predecessor of serverless computing, was more extensively studied. Despite their similarities, their main difference is in their roles. Microservices is a design technique for applications, whereas serverless functions are the architecture for running parts or entire applications. In particular, microservices can be hosted within a serverless architecture. Toledo et al. [13] interviewed 25 practitioners working with microservices in seven large companies. They identified 16 architectural TD issues, their negative impacts, and common solutions to repay each debt and its related costs. Pigazzini et al. [14] studied a real-life project involving approximately 1,000 services within a large international company, quantitatively analyzing the documentation and conducting interviews. The research studies discussed above mainly focus on investigating TD in microservices through expert interviews. On the contrary, our emphasis is on serverless architecture, rather than microservices, and on analyzing TD through the mining of SO questions.

Summary. Several studies have explored various aspects of TD using SO. **Our study exclusively focuses on TD within the context of questions on serverless on SO**. Moreover, in previous work, expert knowledge was used to gain insight into TD in serverless architectures, with a focus on three debt types (code, test, and architecture debt), **whereas our study goes a step further by exploring additional TD types using SO as the knowledge base**. These two key differences allow our study to provide evidence-based insights into the prevalence of TD, serverless-specific challenges encountered by the community, and the availability of solutions. In addition, architectures such as microservices have received extensive research on different types of smells and debt, and on how they are manifested. In our study, we aim to understand how TD manifests in serverless systems. These findings may inform research and practice by highlighting common issues, identifying gaps in existing solutions, and guiding the development of effective tools and practices to address TD in serverless computing.

III. Methodology

A. Goal & Research Questions

The *goal* of this study is to investigate the prevalence, types, specific issues, and solutions of serverless-related questions on SO, with the *purpose* of characterizing TD in serverless computing and identifying the solutions proposed by practitioners. The *perspective* is of both researchers and practitioners. The former are interested in gaining a deeper understanding of TD in serverless computing, with the aim of identifying requirements that may lead to the development of novel automated approaches better suited to practitioners' needs. The latter is interested in understanding how TD manifests and the strategies other practitioners use to address it, with the aim

[1]https://doi.org/10.5281/zenodo.17381567

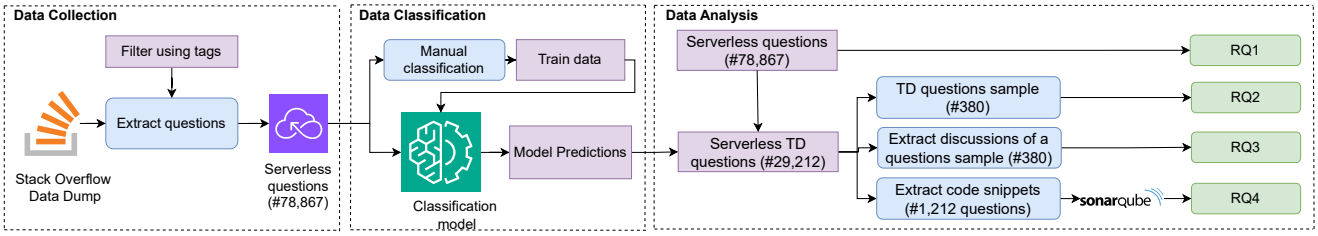

Fig. 1. Study Design

of improving the quality assurance mechanisms in their own contexts.

More specifically, we pose four Research Questions (RQs):

RQ1: To what extent do SO questions in the serverless domain indicate TD? *Rationale:* We classify TD questions into various types and subtypes in order to understand which TD types are most often explored on SO. We aim to compute the proportion of TD present in serverless-related questions on SO and investigate how SO users discuss serverless-related queries. This preliminary analysis can provide insight into the common types and subtypes of TD in serverless environments, helping raise awareness of the issue's practical relevance.

RQ2: What are the types of TD in serverless computing? *Rationale:* We categorize the classified TD questions into different types and subtypes to understand which types of TD are most commonly discussed on SO. The objective is to gain an understanding of the prevailing types of TD discussed by the SO community and identify subtypes specific to each type, which may be instrumental in prioritizing future research and informing practitioners about the types of TD most frequently encountered in practice.

RQ3: How is TD in serverless mitigated? *Rationale:* We aim to understand how serverless TD questions are addressed, providing insights into the solutions practitioners propose to deal with TD in serverless architectures. We analyze the answers to serverless TD questions to identify the types of answers available. We also analyze the availability of answers and accepted answers (resolved problems) for each TD type.

RQ4: To what extent do serverless SO code snippets contain TD? *Rationale:* In addition to analyzing question text to investigate TD in the previous RQs, we bring another perspective using static code analysis. We analyzed code snippets extracted from the collected serverless questions to identify code smells. Code smells are common indicators of TD, which is defined as surface-level indications of underlying issues in the codebase [15], [16], [17]. This analysis will help us understand the types of code smells present in serverless code snippets. Hence, it deepens the understanding of specific quality issues practitioners face when developing serverless applications.

Figure 1 outlines our methodology, which includes three main phases: data collection, classification, and analysis. Initially, we extracted serverless-related questions from SO and then classified them as TD-related or not. Subsequently, we analyzed the TD-related questions to address our RQs. Our methodology adhered to established guidelines by Wohlin et al. [18] and the ACM/SIGSOFT empirical standards[2].

B. Data Collection

We extracted our data from the publicly available Stack Exchange Data Dump[3], which provided a dataset of all questions on the site from January 2014 to March 2025. Quite noteworthy is the fact that the year 2014 was significant in that it was the year of releasing the concept of serverless computing in an "AWS re:Invent' event"[4], thus bringing it into the world. Therefore, we have considered a reasonable time period for data collection that accurately reflects the introduction of serverless architecture to the world.

To ensure accurate retrieval of SO questions, we selected the tags most commonly used by the SO community to extract serverless questions. This was done by analyzing existing research on serverless, which used terms that implied the serverless paradigm. Wen et al. [8] extracted a total of 13 tags as related to serverless computing. These tags included *serverless-architecture, serverless, serverless-framework, aws-serverless, aws-lambda, aws-sam, aws-sam-cli, serverless-offline, vercel, serverless-plugins, localstack, faas,* and *openwhisk*. We used the same tags and expanded the list based on existing studies. Two additional tags, *azure-functions* and *google-cloud-functions*, were identified as related to the concept of serverless, since a few studies [3], [2], [19] reported on these frameworks as serverless computing services delivered by major cloud providers, similar to AWS Lambda. Therefore, we selected 15 tags to extract serverless-related posts. This curation and filtering yielded a substantial dataset comprising 78,867 SO questions on serverless computing.

C. Data Classification

The data classification stage consisted of two phases: manual classification and automated classification. We first manually classified serverless SO questions into two categories (TD-related and non-TD-related). Next, we used the classified questions as a training dataset for a binary text classification model we built in the next phase. Manual classification was done by two annotators, namely (i) the first author, who has

[2]https://github.com/acmsigsoft/EmpiricalStandard
[3]https://archive.org/details/stackexchange
[4]https://aws.amazon.com/blogs/developer/aws-reinvent-2014-recap-2/

4 years of software development experience and 2 years of research experience in software engineering, and (ii) a final year PhD student who has 6 years of research experience in software engineering. We evaluated each question against the Dagstuhl TD definition [20] to determine whether each related to TD. In this stage, the annotators individually classified a sample of 150 serverless SO questions as TD-related or non-TD-related. Cohen's Kappa interrater agreement was 0.83. They discussed and resolved all the disagreements. Then, the annotators continued manually classifying the dataset until they identified 406 TD-related questions. They selected a similar number (419) of non-TD-related questions to create the training dataset. A similar number of non-TD questions were selected to create a balanced dataset for training the binary text classifier. This resulted in a total of 825 SO questions as the training dataset.

We then followed the steps of Kozadinis et al. [9] to develop a binary text classification system. The steps involved: (i) pre-process the collected questions related to the serverless paradigm, (ii) develop a binary text classification model to identify TD questions on the serverless paradigm, (iii) use the classification model to classify the extracted serverless SO questions as related to TD or not.

First, we removed duplicate questions from the dataset. However, we did not find any duplicate questions. Hence, we proceeded to preprocess the questions related to serverless concepts collected from users, ensuring the data was formatted correctly. We removed punctuation and converted all text to lowercase. Then, we removed SO-related HTML or markdown elements and appearances of source code, as these do not contain any information of lexical or semantic importance.

Subsequently, we trained five models, consisting of four traditional machine learning models used by Kozanidis et al. [9] for the binary text classification of SO questions, Logistic Regression, Random Forest classifier [21], Support Vector Machine (SVM), Naive-Bayes classifier, and a deep learning model, BERT [22], to identify serverless TD questions and adhere to the guidelines established. BERT was selected because it is an advanced transformer-based model suitable for text classification. Our classification models were evaluated using standard metrics consistent with established practices, namely precision, recall, and the F1-score. During the model training phase, the models achieved the following F1-score: fine-tuned BERT 0.86, Logistic Regression 0.81, Random Forest classifier 0.82, Naive-Bayes classifier 0.81, and SVM 0.78. The model with the highest F1-score was selected to predict the class of the unseen data, specifically the serverless SO questions collected during the data collection phase. We use the collected serverless SO questions to conduct an in-depth analysis of TD- and non-TD-related questions on SO and to gain insights into various aspects of serverless questions.

D. Data Analysis

During the final phase of the study, we analyzed the serverless TD questions to identify the prevalence of TD, its types, how TD is mitigated, and whether TD occurs in the code snippets of SO questions.

To investigate the prevalence of TD (RQ1), we built a classification model as explained Section III-C. For further analysis of the types of TD in serverless (RQ2), we followed a content analysis approach. There are different taxonomies for TD classification [23], [24], [25], [26], [27]. We used the TD type taxonomy presented by Li et al. [27] to categorize each selected TD question in the sample into ten types: *architecture, build, code, defect, design, documentation, infrastructure, test, requirement,* and *versioning*. We first selected a statistically significant sample (95% confidence and 5% margin of error out of 29,212 serverless TD questions) of 380 questions. During the manual categorization process, we identified one additional TD type, *security debt*, in addition to the 10 TD types included in the taxonomy by Li et al. [27]. Therefore, in total, we considered 11 types of TD when categorizing the serverless TD question sample (of 380 questions). We then used open coding [28] to identify the subtypes for each TD type. We first broke the question text into smaller parts during the open coding process to identify key information related to a particular serverless computing problem. Then, using the information we extracted from the text, we assigned each question a "code." We proceeded in the same manner with the subsequent questions. We classified the questions into one of the "codes" we had already identified; if not, we created a new "code." This resulted in 49 codes. One author conducted the analysis by categorizing the SO questions into types and subtypes. To reduce bias, a random sample of 50 questions was independently assessed by the PhD student (the second annotator in Section III-C). Then, we used Cohen's Kappa [29] to calculate the inter-rater agreement. This yielded a value of 0.94, indicating near-perfect agreement among the raters.

To investigate the proposed solutions to the serverless TD questions (RQ3), we conducted a qualitative analysis of a sample of 380 questions to further identify the problems in serverless TD questions and how they have been addressed. To achieve this, we reviewed the complete discussions of the 380 SO questions, including the answers. We used open coding to categorize the solutions by availability and type. For each TD type, we examine the availability of solutions and the accepted solutions to identify those that require critical attention.

To investigate the presence of TD in code snippets (RQ4), we extracted code snippets from the subset of serverless-related SO questions. We used SonarQube[5] to analyze the extracted code snippets and identify code smells present. Even though we did not expect the tool to detect serverless-specific smells, we investigated whether it could detect at least indicators of such smells, such as the bad practices in serverless application development identified by Taibi et al. [2]. Eskandani et al. [30] found that the most commonly used languages for developing serverless applications are JavaScript, TypeScript, and Python. As such, we only extracted code snippets written in these languages. Therefore, we extracted code snippets for

[5]https://www.sonarsource.com/products/sonarqube/

1,212 SO questions. We selected SonarQube as the static code analysis tool due to its popularity in the literature for detecting TD in code [31], [32], [33]. Using SonarQube, we evaluated the security, maintainability, and reliability. We selected these three metrics because similar studies [32], [34] have used them to detect TD, code smells, and security vulnerabilities in source code, which we aim to investigate under RQ4. According to the SonarQube documentation[6], security refers to protecting software from unauthorized access, use, or destruction. Maintainability refers to how easy it is to repair and improve code, and reliability refers to the probability of a system performing its required functionality without failure [35]. Regarding maintainability issues, we consider code smells, TD, and TD ratio metrics. SonarQube defines the TD metric as the sum of the remediation costs for the maintainability issue. Issue remediation cost refers to the effort (in minutes) required to fix the issue. The TD ratio is the ratio between the cost to develop the software and the TD (the cost to fix it).

IV. Results

A. RQ1: To what extent do SO questions in the serverless domain indicate TD?

We used the BERT model [22] to classify the serverless-related questions we extracted from SO as TD- or non-TD-related, which showed the highest F1-score (0.86) among the five models we trained. We used pre-defined weights to train the model and fine-tuned it for our classification task. Our model identified 29,212 of 78,867 serverless-related questions on SO as TD-related, representing 37% of the serverless questions on SO. This significant proportion highlights the pressing need for additional research in the serverless domain. Nonetheless, it is worth noting that the BERT model's F1-score was 0.86. The validity of the classification results depends on the model's accuracy. Therefore, the percentage presence of TD in serverless may differ slightly from 37%. To mitigate these threats, we manually classified a statistically significant sample (95% confidence, 5% margin of error), resulting in 383 SO questions. We used it to evaluate the model's performance. This resulted in an F1-score of 0.82, which is close to the model's F1-score.

Example I: *"Anyone know how to set a websocket's content handling strategy to binary in the serverless framework? I have a websocket defined as follows: [code] I'm looking for a sls approach since all of my infra is managed in sls and I don't want to absorb the* ***tech debt*** *migrating (unless I have to)"*

Example II: *"Our Google Analytics data events are exported to BigQuery tables. I have reports that need to run when the events data arrives which are set up as AWS lambdas with python code (for various reasons and I can't immediately move these to be Google Cloud Functions etc). Is it possible to have the creation of a table trigger a lambda? At present, I have a lambda periodically checking to see if the table has been created which seems* ***suboptimal****. Eventarc looks like it might possibly be the way to monitor for the creation event at the BigQuery end but it doesn't seem obvious how you'd interface with AWS. Any genius ideas? I have dug repeatedly through StackOverflow, but can't see a match for this issue"*

As shown in Examples I and II (verbatim), SO users explicitly discuss TD-related queries and use terms such as "suboptimal" to refer to TD. In both examples, users are seeking better solutions to the problems they encounter. The results of our classification model indicate that 37% of serverless questions on SO are related to TD. This provides a brief insight into the prevalence of TD in serverless computing. When SO users ask questions on the platform, it is expected that a certain number of those questions will translate to TD [9], [10]. Also, it should be noted that the number of serverless questions that depict TD is significant and needs further investigation.

RQ1 Summary: Out of the 78,867 serverless questions on SO, 37% were identified as TD related questions. SO users discuss TD-related questions explicitly using words such as "tech debt" and others that indicate TD, such as "suboptimal."

B. RQ2: What are the types and subtypes of TD in serverless computing?

We followed the taxonomy defined by Li et al. [27] to categorize the TD questions into TD types. We manually categorized a sample of 380 questions into 11 TD types.

According to Table I, the most prevalent type of TD is *code debt*. This is followed by *design debt*, the second-most prevalent type of TD, and *versioning debt*, the third-most prevalent. The least common type of TD in serverless SO questions is *infrastructure debt*. However, we did not find any questions related to *requirement debt*. Therefore, based on RQ2, we identified 10 types of TD and their prevalence in serverless computing. Table I also includes a reference to an example of each type of TD, which can also be found in the replication package, and the number of TD instances for each type.

We looked further into the problems that occur in serverless TD questions. As depicted in Table I, we identified the problems associated with each type of TD. We used open coding to categorize the TD types further based on the nature of the issue. This resulted in 49 codes. We identified six subtypes of *architecture debt* issues, with those related to architecture selection being the most common. For *build debt*, deployment-related issues were the most prevalent. *Code debt* encompasses a range of issues, from general implementation concerns to data transfer between functions. We identified four subtypes for *defect debt*. Most of the *design debt* issues were related to function timeouts. Lack of documentation and incorrect documentation were the subtypes of *documentation debt*. *Infrastructure debt* had five subtypes, with scaling being the most common. Authentication and authorization-related issues were the most common among *security debt*. *Test debt*

[6]https://docs.sonarsource.com/sonarqube/latest/user-guide/clean-code/software-qualities/

TABLE I
TYPES AND SUBTYPES OF TD

Type	Sub Type	Count	Total
Architecture[7]	Scaling the application	4	40
	Architecture selection	11	
	Architecture change	6	
	Architecture implementation	9	
	Cold start	5	
	Cost related	5	
Build[8]	Deployment related	18	28
	Build configurations	10	
Code[9]	Database CRUD operations	22	129
	Function I/O	1	
	Add/access logs	6	
	Invoking functions	19	
	Handling media files	11	
	Data transfer between services	21	
	Invoking external APIs	8	
	Handling events	4	
	Connection between services	14	
	Concurrent executions	8	
	Refactoring	1	
	Implementation	14	
Defect[10]	Clearing data	6	24
	Query optimization	4	
	View logs	2	
	Function specific	11	
Design[11]	Service specific	12	49
	Function timeout	14	
	Size limitations	2	
	Design patterns	5	
	Invoking functions	8	
	Scheduling tasks	5	
	Directory structure	2	
	Shared services	1	
Documentation[12]	Lack of documentation	15	15
Infrastructure[13]	Scaling	4	8
	Minimizing cost	1	
	Building infrastructure	1	
	Memory limitations	1	
	Monitoring services	1	
Security[14]	Secrets management	7	31
	Authentication and authorization	15	
	Accessing other services	3	
	Limiting access	6	
Test[15]	Testing locally	3	9
	Testing deployed functions	3	
	Mocking API calls	3	
Versioning[16]	Upgrading the version	35	47
	Downgrading the version	2	
	Version selection	2	
	Version capabilities	4	
	Version deprecation	4	

reported issues in testing locally, testing deployed functions, and mocking API calls. The most common subcategory of *versioning debt* relates to upgrading to a new version. To identify issues specific to serverless computing, we compared our results with the existing literature and identified 6 serverless-specific issues. This is further discussed in Section V-A.

TABLE II
PERCENTAGE OF QUESTIONS WITHOUT ANSWERS & ACCEPTED ANSWERS

TD Type	Percentage of Questions without Answers	Percentage of Questions without Accepted Answers
Architecture	12.5	67.5
Build	17.9	**75.0**
Code	12.4	51.2
Defect	25.0	62.5
Design	20.4	69.4
Documentation	**53.3**	66.6
Infrastructure	12.5	50.0
Security	12.9	48.4
Test	33.3	66.6
Versioning	14.9	41.1

RQ2 Summary: The most prevalent type of TD in serverless-related questions was *code debt*, with *design debt* and *versioning debt* as the second and third common types of TD, respectively. The least common type of TD was *infrastructure debt*. We identified 49 TD subtypes in serverless SO questions.

C. RQ3: How is TD in serverless mitigated?

To understand how SO users have answered the TD-related questions, we categorized the solutions into four main types: 1) not answered, 2) not possible, but an alternative was suggested, 3) not possible, and 4) a solution was provided. For the questions with provided solutions, we further broke them down into six categories, as shown in Table III. We categorized the answers to serverless questions, but noticed that the solution categories we identified were not specific to serverless and can be applied to SO questions from other domains. However, we present the results from a serverless perspective. We further investigated the availability of answers and accepted answers for each TD type.

Table II shows the percentage of questions without answers and the percentage of questions without accepted answers for each type of serverless TD questions. *Documentation debt* has the highest percentage (53.3%) of unanswered questions, indicating it needs more attention from the SO community. *Code debt* questions have the lowest percentage (12.5%) of unanswered questions, indicating that the majority are answered on SO. Based on the absence of accepted answers, *build debt* had the highest percentage, with 75% of questions lacking accepted answers. *Versioning debt* has the lowest percentage (41.1%) of questions without accepted answers.

[7]https://stackoverflow.com/questions/63427991
[8]https://stackoverflow.com/questions/61475946
[9]https://stackoverflow.com/questions/75950215
[10]https://stackoverflow.com/questions/60709896
[11]https://stackoverflow.com/questions/69471022
[12]https://stackoverflow.com/questions/74287020
[13]https://stackoverflow.com/questions/42234413
[14]https://stackoverflow.com/questions/59653206
[15]https://stackoverflow.com/questions/50308205
[16]https://stackoverflow.com/questions/73860735

TABLE III
SOLUTIONS OF SERVERLESS TD QUESTIONS

Solution Availability	Solution Type	Definition	Count
Solution provided	Code modifications	Implementation of the solution is provided along with the code snippets.	106
	Alternative solution	The solution suggested in the question is incorrect. Therefore, an alternative method is suggested.	14
	Possible suggestions	Multiple suggestions to approach the problem are suggested as it is unable to find the exact solution with the given information.	16
	Debugging and troubleshooting	Clear guidelines, including specific instructions to pinpoint and solve the problem, are suggested.	13
	Conceptual understanding	The solution is comprised of a detailed explanation of the concept asked in the problem.	37
	Knowledge resources	Official documentation or implementation of the solution to a similar problem is referenced.	84
	Tool suggestions	Suggested the use of a tool to solve the problem.	3
	Best practices	Provided suggestions based on general guidelines or industry standards.	6
	Configuration changes	Suggested adjustments to the environment, services, or tools.	21
Not possible		It is impossible to achieve a solution to the problem due to the limitations of the services.	5
Not possible, alternative given		It is impossible to achieve a solution, but an alternative approach to solving the problem is suggested.	8
Not answered		A solution has not been provided.	67

RQ3 Summary: The analysis of the types of TD in serverless computing revealed various issues within each category of TD. *Documentation and build debt* were identified as types of TD with the highest percentages of unanswered and unresolved questions. Notably, 78% of the questions received responses from SO users, while 18% remained unanswered or posed unsolvable challenges. As potential solutions to the question asked, SO users provided implemented code, detailed explanations, multiple possible solutions, step-by-step guidelines, and references to examples.

D. RQ4: To what extent do serverless SO code snippets contain TD?

We investigated code smells and TD in code snippets extracted from 1,212 SO questions. We ran SonarQube on 69k lines of code, focusing on three types: maintainability, reliability, and security vulnerabilities, as these were the only metrics relevant to our study. The analysis identified 8 security vulnerabilities, 949 code smells related to maintainability, and 51 code smells related to reliability. Table IV summarizes the results of the SonarQube analysis. The detected security vulnerabilities were high-severity. TD and TD ratio measurements for maintainability were 25 days and 1.0%, respectively. Most reliability code smells were of low severity, whereas most maintenance code smells were of medium and low severity.

Listed below are a few examples of the maintainability and reliability code smells detected by the static analysis tool.

- Unexpected var, use let or const instead.
- Remove this unused import.
- Remove the unused variable.
- Do not import modules using an absolute path.

TABLE IV
TD IN CODE SNIPPETS

Type	Count	Severity		
		Low	Medium	High
Security vulnerabilities	8	0	0	8
Reliability code smells	51	46	2	3
Maintainability code smells	949	420	395	134

- Don't use 'datetime.datetime.utcnow' to create this datetime object.
- Remove this assignment to a local variable; the value is never used.
- Remove this commented out code.
- Import only needed names or import the module and then use its members.

Below are examples of security vulnerabilities identified.

- Make sure this private key gets revoked, changed, and removed from the code.
- Don't use a JWT token without verifying its signature.

This is further discussed in Section V-B.

RQ4 Summary: SonarQube detected 949 code smells related to maintainability issues, 51 reliability code smells, and 8 security vulnerabilities in the extracted code snippets of serverless SO questions. Most of the maintainability issues were medium- or low-severity, the majority of reliability issues were low-priority, and all the security vulnerabilities were high-priority.

V. Discussion

A. Serverless Specific Issues

In serverless architectures, certain TD issues emerge that are unique to serverless architectures. These issues often stem from challenges in developing serverless applications, particularly in performance, operations, and developer experience [36]. In Section IV-B, we identified types and subtypes of TD that are present in serverless-related SO questions. However, most of these issues (e.g., CRUD database operations, upgrading versions, invoking external APIs, etc.) also apply to other architectures. Therefore, among the TD issues we identified in Section IV-B below, we elaborate on those specific to serverless. Detailed examples of questions of each TD type can be found in the replication package.

Scaling the Application. Serverless computing has an auto-scaling feature where the necessary resources are allocated based on the demand [37]. In serverless computing, scaling is handled automatically by the cloud provider. This abstracts away the infrastructure management from the developer [38], [39]. However, this can lead to unexpected challenges. For instance, serverless functions may face limitations in scaling, especially when dependent services or resources do not scale at the same rate. This issue is unique to serverless because, in traditional architectures, scaling typically involves scaling entire server instances, which developers can directly manage and optimize. This finding aligns with the study of Baldini et al. [6], which identified scaling as an open problem in serverless.

Cold Start. Cold start latency is a well-known issue in serverless computing. This occurs when a function is invoked after an idle period, resulting in a delay. This latency can introduce unexpected delays in function startup, thereby negatively affecting performance [40], [41], [42], [43], [44], [45], [46]. Unlike traditional architectures, where servers are always running and readily available, serverless functions are stateless and may require initialization before execution, which can lead to cold-start issues. This finding corroborates those of Baldini et al. [6] and Wen et al. [8], who identify calling to zero as a key differentiator in serverless, which ultimately leads to cold-start problems.

Add/Access Logs. Logging in serverless computing refers to the logs produced when a serverless function is run. This can be used to monitor and debug the applications. Serverless functions often run in ephemeral environments, making traditional logging practices more challenging. Accessing and aggregating logs from these short-lived instances can be cumbersome, needing specialized tools or services [47]. This contrasts with traditional server-based architectures, where logs are typically stored on persistent disks and can be accessed directly. The ephemeral nature of serverless functions requires a rethinking of logging strategies, often leading to TD when not properly managed. Baldini et al. [6], Lenarduzzi et al. [3], and Wen et al. [8] identified similar problems in monitoring and debugging serverless functions, and inadequate or verbose logging, respectively.

Minimizing Cost. Cost optimization is both a benefit and a challenge in serverless computing. The pay-per-use model means that costs can quickly escalate if the functions are not optimized for execution time and resource usage [48], [49]. Unlike traditional server-based applications, serverless costs are more dynamic and require constant monitoring and optimization. This can lead to TD if costs are not carefully considered during the design and development phases. Our findings align with those of Baldini et al. [6], who identified cost as a fundamental challenge in serverless.

Secrets Management. Managing sensitive information, such as API keys or credentials, is more complex in serverless architectures. The stateless nature of serverless functions means that secrets cannot be stored in the same way as in stateful servers. This can introduce security vulnerabilities if handled incorrectly, making secrets management a critical and unique issue in serverless computing. Lenarduzzi et al. [3] identified managing secrets as a cause of TD, complementing the findings of our study.

Testing Locally. Testing serverless functions locally can introduce significant challenges due to the tightly coupled nature of these functions with cloud services. Simulating the cloud environment locally can be challenging and lead to discrepancies between local test results and actual behavior in production [50]. This issue is specific to serverless due to its integration with managed cloud services, which are not easily replicated in a local testing environment. Lenarduzzi et al. [3] identified testing serverless functions as a difficult task that can accrue TD if not done properly and on time.

Baldini et al. [6] identified issues related to cold start, scaling, accessing logs, and minimizing cost, whereas Wen et al. [8] identified issues related to cold start and accessing logs. However, our analysis focused on the TD perspective, while theirs concentrated on identifying common issues in serverless. Our work demonstrates that, if unresolved, the aforementioned issues gradually accrue TD. Through expert interviews, Lenarduzzi et al. [3] determined that testing locally, secret management, and accessing logs are activities that generate TD. Our research is complementary, offering empirical support from a broader perspective (SO users).

B. TD in Serverless SO Code Snippets

To better understand TD in serverless applications, we analyzed code snippets extracted from SO questions on Serverless using SonarQube to detect code smells. We found several common code smells, but they are largely generic to any software architecture. However, by comparing the code smells identified by SonarQube (listed under Section IV-D) with the issues identified under Section IV-C, we found that although SonarQube detected code smells, it was unable to identify issues specific to serverless.

Taibi et al. [2] introduced everal serverless bad practices, including excessive use of synchronous functions, high use of libraries, presence of shared code, and more than one technology employed. However, in our study, a review of code smells found by SonarQube and a comparison of the results

in Sections IV-D and IV-C reveal that the tool fails to identify these serverless-specific bad practices.

SonarQube's inability to detect serverless-specific TD indicates a gap in code quality analysis tools currently available. As serverless computing gains more traction, the demand for tools that detect serverless-specific problems is required. These tools would need to focus on serverless-specific features.

C. Availability of Solutions to Serverless TD

The pattern in the number of answers and accepted answers for serverless-related questions shows that TD questions are more likely to remain unanswered than non-TD questions. Out of 29,212 TD-related questions, 21% remain unanswered, and 40% have no accepted answers. To compare, out of 49,655 non-TD questions, 19% remain unanswered and 40% have no accepted answers.

Even though there may be several reasons behind unanswered questions [51], [52], the high percentage of unanswered TD questions signifies limited knowledge and expertise in this area. It raises questions about conducting further research, involving the community, and providing better resources to help developers handle TD in serverless systems. As serverless computing evolves, the community's ability to offer end-to-end solutions to such TD-related problems will be determinant to the continued success of serverless computing.

D. Code Debt in Serverless Computing

For RQ3, we categorized the serverless TD questions into 11 TD types. We identified that *code debt* is the most common type of TD. Code debt refers to poorly written code that deviates from best practices [27]. However, this might be because serverless-related tags are often used along with other tags. For the study, we collected all SO posts that had one of the selected relevant tags. Therefore, the selected posts may have multiple tags, including those that do not belong to serverless.

TABLE V
DISTRIBUTION OF SERVERLESS-RELATED TAGS

Tag	Count (Anywhere)	Count (Alone)	Percentage (%)
aws-lambda	13211	125	0.95
azure-functions	6685	178	2.66
google-cloud-functions	5467	54	0.99
serverless	1786	9	0.50
serverless-framework	1570	51	3.25
vercel	1436	29	2.02
aws-serverless	616	2	0.32
aws-sam	526	9	1.71
aws-sam-cli	237	3	1.27
localstack	156	3	1.92
serverless-architecture	115	3	2.61
openwhisk	60	5	8.33
serverless-offline	54	0	0.00
faas	25	0	0.00
serverless-plugins	23	0	0.00

Table V shows a breakdown of the number of questions per tag with that tag as the only tag in the posts (count (alone)) and the number of questions that included that tag in the post, but also included other tags (count (anywhere)), and the percentage of tags count (alone) of the count (anywhere). If we consider the tag "aws-lambda," only 0.95% of all questions have it as the only tag. This may explain why the percentage of *code debt* in serverless TD questions is higher than in other TD types. Therefore, it is evident that SO users discuss serverless-related TD questions using many other tags besides those specifically related to serverless.

VI. IMPLICATIONS

For Researchers. Our study points out various serverless-specific challenges that offer a rich area for exploration. Common types of TD present in serverless architecture, such as the accumulation of TD in code, design, and security, can be studied in greater detail by examining serverless applications to understand how they appear in serverless code. In addition, further research can be conducted to explore how different types of TD evolve over time, how they impact application performance and code maintainability, and how effectively different solutions resolve them. Based on the findings of this research, researchers can develop more robust frameworks and guidelines for addressing TD in serverless computing.

For Practitioners. Practitioners can utilize our findings to understand how TD in serverless computing is being tackled. By identifying the most commonly encountered types and subtypes of TD and their solutions, our research can serve as a reference for developers to identify similar problems in their applications and fill those gaps. Senior developers can leverage these insights to refine their strategies for managing TD, and junior developers can use this experience to better deal with the complexities of serverless computing in production. Our findings can help practitioners learn from shared mistakes and make informed decisions when employing serverless technology.

For Tool Vendors. Static analysis of serverless code fragments showed a gap in existing tools, i.e., the inability to detect TD or serverless-specific smells. SonarQube, while effective against generic code smells, lacks the capability to detect serverless-specific smells. It can be leveraged by tool vendors to develop tools specifically meant for use in serverless environments. By creating products that effectively identify and manage serverless-specific TD, tool vendors enable developers to achieve improved code quality and save costs in the long term. This would not only benefit the development community but also put vendors at the forefront of an emerging technology.

VII. THREATS TO VALIDITY

We identify the threats to the validity of our study following the classification by Runeson et al. [53].

Internal validity is the degree to which the "treatment," not other variables, is responsible for the observed results. The classification methodology used to detect serverless TD questions poses the greatest threat to the internal validity of our study. The basis for training the classification model was

the BERT model [54], achieving an F1-score of 0.86. It's important to note that potential inaccuracies in the dataset might influence results, leading to false positives and negatives. However, for automatic TD classification, we trained multiple models and selected the one with the highest accuracy. In addition to BERT, we trained four models and selected the one with the highest F1-score. Also, we manually classified a statistically significant sample of questions as TD-related or not. We then evaluated the model on the sample, achieving an F1-score of 0.82. Another threat to the internal validity of our study is the tool we used to analyze the code snippets. We are aware that many other static code analysis tools can be used to detect TD and code smells. However, we believe that SonarQube is the most commonly used tool for static code analysis, based on multiple studies [31], [32], [33]. Using SO as the knowledge base poses another threat to the internal validity of our study. Although LLMs like ChatGPT[17] have made SO less popular, we still consider it an invaluable resource for software engineering research. 62% of the code produced by LLMs contains API misuse, according to a recent study by Zhong et al. [55] that compared LLMs and SO on coding questions. According to another study by Kabir et al. [56], 72% of ChatGPT responses were verbose, and 52% contained inaccurate information. Therefore, we regard SO to have higher reliability and precision in answering SO questions, as demonstrated by recent studies, even though we acknowledge developers' use of LLMs like ChatGPT when developing software.

External validity regards the generalizability of our study. We conducted the study using SO questions. However, there are other platforms such as GitHub[18] and SE Stack Exchange[19] which can be used for the same purpose. Therefore, we cannot generalize our results. However, as mentioned in the study by Kozanidis et al. [9], with over 24 million questions and 20 million registered users, SO is the most popular programming Q&A website among the well-known group of Q&A websites. We consider SO the best data source for developers to discuss issues related to TD and serverless software development, given its size and popularity.

Construct validity pertains to the extent to which our operational measures are appropriately aligned and capable of addressing our RQs. We extracted the data using the SO tags that we identified. There can be errors when SO users add tags to the questions. To mitigate this threat, we manually validated the selected sample in Section III-B to ensure the questions were related to the mentioned tags.

Conclusion validity relates to how our treatments and outcomes relate to one another. Using a set of tags we determined to be relevant to serverless, we extracted questions about serverless. The results of our study will be affected by the possibility that additional tags are associated with serverless than the eight we examined. Furthermore, we obtained F1, precision, and recall scores of 0.86 for our classification model. This implies that there may have been false positives and negatives in our model, which could have affected our outcomes. Another threat is the manual categorization of the TD questions. Manual categorization may introduce bias and subjective judgment for the analysis. To reduce bias, we asked a graduate student to independently classify a sample of questions, and the interrater agreement was 0.94 (Cohen's Kappa).

[17]https://chatgpt.com/

[18]https://github.com/

[19]https://softwareengineering.stackexchange.com/

VIII. Conclusions and Future Work

Our analysis revealed that 37% of SO's serverless questions are related to TD, and in the majority of discussions, TD is either explicitly or implicitly mentioned. Code debt was the most prevalent type, followed by design and versioning debts. We also revealed common code smells and security vulnerabilities in the code samples of these discussions. These results illustrate new challenges in serverless architecture, expose current research limitations, and indicate the urgent need for expert analysis tools. In the future, we plan to study other platforms, such as GitHub issues, improve our classification model, and develop a static analysis tool for serverless-specific code smells to aid TD mitigation in serverless.

References

[1] M. Cinque, "Real-time faas: serverless computing for industry 4.0," *Service Oriented Computing and Applications*, vol. 17, no. 2, pp. 73–75, 2023.

[2] D. Taibi, B. Kehoe, and D. Poccia, "Serverless: From bad practices to good solutions," pp. 85–92, 2022.

[3] V. Lenarduzzi, J. Daly, A. Martini, S. Panichella, and D. A. Tamburri, "Toward a technical debt conceptualization for serverless computing," *IEEE Software*, vol. 38, no. 1, pp. 40–47, 2021.

[4] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 47–52. [Online]. Available: https://doi.org/10.1145/1882362.1882373

[5] M. H. Tanzil, S. Chowdhury, S. Modaberi, G. Uddin, and H. Hemmati, "A systematic mapping study of crowd knowledge enhanced software engineering research using stack overflow," *Journal of Systems and Software*, p. 112405, 2025.

[6] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski *et al.*, "Serverless computing: Current trends and open problems," *Research advances in cloud computing*, pp. 1–20, 2017.

[7] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of function-as-a-service software development in industrial practice," *Journal of Systems and Software*, vol. 149, pp. 340–359, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218302735

[8] J. Wen, Z. Chen, Y. Liu, Y. Lou, Y. Ma, G. Huang, X. Jin, and X. Liu, "An empirical study on challenges of application development in serverless computing," in *Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2021, pp. 416–428.

[9] N. Kozanidis, R. Verdecchia, and E. Guzman, "Asking about technical debt: Characteristics and automatic identification of technical debt questions on stack overflow," p. 45–56, 2022. [Online]. Available: https://doi.org/10.1145/3544902.3546245

[10] J. A. Edbert, S. J. Oishwee, S. Karmakar, Z. Codabux, and R. Verdecchia, "Exploring technical debt in security questions on stack overflow," in *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2023, pp. 1–12.

[11] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun, "What security questions do developers ask? a large-scale study of stack overflow posts," *Journal of Computer Science and Technology*, vol. 31, pp. 910–924, 2016.

[12] F. Tian, P. Liang, and M. A. Babar, "How developers discuss architecture smells? an exploratory study on stack overflow," pp. 91–100, 2019.

[13] S. Toledo, A. Martini, and D. Sjøberg, "Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study," *Journal of Systems and Software*, vol. 177, p. 110968, 04 2021.

[14] I. Pigazzini, F. A. Fontana, V. Lenarduzzi, and D. Taibi, "Towards microservice smells detection," p. 92–97, 2020. [Online]. Available: https://doi.org/10.1145/3387906.3388625

[15] D. Das, A. A. Maruf, R. Islam, N. Lambaria, S. Kim, A. S. Abdelfattah, T. Cerny, K. Frajtak, M. Bures, and P. Tisnovsky, "Technical debt resulting from architectural degradation and code smells: a systematic mapping study," *ACM SIGAPP Applied Computing Review*, vol. 21, no. 4, pp. 20–36, 2022.

[16] J. R. Lahti, A.-P. Tuovinen, and T. Mikkonen, "Experiences on managing technical debt with code smells and antipatterns," in *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 2021, pp. 36–44.

[17] A. Arif and Z. A. Rana, "Refactoring of code to remove technical debt and reduce maintenance effort," in *2020 14th International Conference on Open Source Systems and Technologies (ICOSST)*. IEEE, 2020, pp. 1–7.

[18] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén *et al.*, *Experimentation in software engineering*. Springer, 2012, vol. 236.

[19] Y. Li, Y. Lin, Y. Wang, K. Ye, and C. Xu, "Serverless computing: state-of-the-art, challenges and opportunities," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1522–1539, 2022.

[20] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering (dagstuhl seminar 16162)," *Dagstuhl reports*, vol. 6, no. 4, pp. 110–138, 2016.

[21] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[22] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[23] D. Pina, A. Goldman, and G. Tonin, "Technical debt prioritization: Taxonomy, methods results, and practical characteristics," in *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2021, pp. 206–213.

[24] N. S. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spínola, "Towards an ontology of terms on technical debt," in *2014 Sixth International Workshop on Managing Technical Debt*. IEEE, 2014, pp. 1–7.

[25] N. S. Alves, T. S. Mendes, M. G. De Mendonça, R. O. Spínola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study," *Information and Software Technology*, vol. 70, pp. 100–121, 2016.

[26] N. Rios, M. G. de Mendonça Neto, and R. O. Spínola, "A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners," *Information and Software Technology*, vol. 102, pp. 117–145, 2018.

[27] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, 2015.

[28] E. Blair, "A reflexive exploration of two qualitative data coding techniques," *Journal of Methods and Measurement in the Social Sciences*, vol. 6, no. 1, pp. 14–29, 2015.

[29] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

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

[31] R. Alfayez, R. Winn, W. Alwehaibi, E. Venson, and B. Boehm, "How sonarqube-identified technical debt is prioritized: An exploratory case study," *Information and Software Technology*, vol. 156, p. 107147, 2023.

[32] V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, "Does migrating a monolithic system to microservices decrease the technical debt?" *Journal of Systems and Software*, vol. 169, p. 110710, 2020.

[33] N. Saarimaki, M. T. Baldassarre, V. Lenarduzzi, and S. Romano, "On the accuracy of sonarqube technical debt remediation time," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019, pp. 317–324.

[34] V. Lenarduzzi, A. Janes, T. Kilamo, V. Mandić, S. Romano, and M. T. Baldassarre, "Improving programming education through static analysis tools using sonarqube: A multiple empirical studies of how students understand rule violations, perceive the criticality of the rule, and the solution difficulty," *Perceive the Criticality of the Rule, and the Solution Difficulty*.

[35] S. U. Farooq, S. Quadri, and N. Ahmad, "Metrics, models and measurements in software reliability," in *2012 IEEE 10th international symposium on applied machine intelligence and informatics (SAMI)*. IEEE, 2012, pp. 441–449.

[36] E. Van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "A spec rg cloud group's vision on the performance challenges of faas cloud architectures," in *Companion of the 2018 acm/spec international conference on performance engineering*, 2018, pp. 21–24.

[37] M.-N. Tran and Y. Kim, "Concurrent service auto-scaling for knative resource quota-based serverless system," *Future Generation Computer Systems*, 2024.

[38] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE international conference on cloud engineering (IC2E)*. IEEE, 2018, pp. 159–169.

[39] G. Somma, C. Ayimba, P. Casari, S. P. Romano, and V. Mancuso, "When less is more: Core-restricted container provisioning for serverless computing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 1153–1159.

[40] L. Yu, L. Fu, and S. Chenhao, "Serverless cold start performance optimization based on multi-request processing and adaptive hierarchical scaling," *IEEE Access*, 2024.

[41] S. Agarwal, M. A. Rodriguez, and R. Buyya, "A reinforcement learning approach to reduce serverless function cold start frequency," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 797–803.

[42] M. Ghorbian and M. Ghobaei-Arani, "A survey on the cold start latency approaches in serverless computing: an optimization-based perspective," *Computing*, pp. 1–55, 2024.

[43] P. Vahidinia, B. Farahani, and F. S. Aliee, "Cold start in serverless computing: Current trends and mitigation strategies," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 2020, pp. 1–7.

[44] P. Verma, P. Goel, and N. Rani, "A review: Cold start latency in serverless computing," in *2024 Sixth International Conference on Computational Intelligence and Communication Technologies (CCICT)*. IEEE, 2024, pp. 141–148.

[45] A. Ebrahimi, M. Ghobaei-Arani, and H. Saboohi, "Cold start latency mitigation mechanisms in serverless computing: taxonomy, review, and future directions," *Journal of Systems Architecture*, p. 103115, 2024.

[46] A. Karamzadeh, A. Shameli-Sendi, and M. Dagenais, "Reducing cold start delay in serverless computing using lightweight virtual machines," *Journal of Network and Computer Applications*, p. 104030, 2024.

[47] J. Manner, S. Kolb, and G. Wirtz, "Troubleshooting serverless functions: a combined monitoring and debugging approach," *SICS Software-Intensive Cyber-Physical Systems*, vol. 34, pp. 99–104, 2019.

[48] Ö. Sedefoğlu and H. Sözer, "Cost minimization for deploying serverless functions," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 2021, pp. 83–85.

[49] P. G. Sarroca and M. Sánchez-Artigas, "Mlless: Achieving cost efficiency in serverless machine learning training," *Journal of Parallel and Distributed Computing*, vol. 183, p. 104764, 2024.

[50] D. De Silva and L. Hewawasam, "The impact of software testing on serverless applications," *IEEE Access*, 2024.

[51] M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A. Schneider, "Answering questions about unanswered questions of stack overflow," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 97–100.

[52] R. K. Saha, A. K. Saha, and D. E. Perry, "Toward understanding the causes of unanswered questions in software information sites: a case study of stack overflow," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, 2013, pp. 663–666.

[53] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, pp. 131–164, 2009.

[54] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *arXiv preprint arXiv:2006.03654*, 2020.

[55] L. Zhong and Z. Wang, "Can llm replace stack overflow? a study on robustness and reliability of large language model code generation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 38, no. 19, 2024, pp. 21 841–21 849.

[56] S. Kabir, D. N. Udo-Imeh, B. Kou, and T. Zhang, "Is stack overflow obsolete? an empirical study of the characteristics of chatgpt answers to stack overflow questions," in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–17.