Journal of Computer and Knowledge Engineering, Vol. 7, No. 2, 2024. (27-42)

Ferdowsi University of Mashhad

Journal of Computer and Knowledge Engineering

https://cke.um.ac.ir

Information and Communication Technology Association of Iran

Evaluating Developers' Expertise in Serverless Functions by Mining Activities from Multiple Platforms [*]

Research Article

Aref Talebzadeh Bardsiri[1], Abbas Rasoolzadegan[2]

DOI: 10.22067/cke.2024.84447.1103

Abstract In the domain of software development, the evaluation of developer expertise has gained prominence, particularly with the rise of serverless functions. These functions, which simplify the development process by delegating infrastructure management to cloud providers, are becoming more common. As developers may utilize functions created by their peers, understanding the expertise of the original developer is crucial since it can serve as an indicator of the functions' quality. While there are existing methods for expertise evaluation, certain gaps remain, especially concerning serverless functions. To address this, our research aims to enhance the assessment of developer expertise in this area by extracting activity-based features from both GitHub and Stack Overflow. After processing the extracted data, we applied various machine learning algorithms. Our findings suggest a potential improvement in evaluating developer expertise when incorporating features from Stack Overflow compared to using only GitHub data. The extent of this improvement was observed to differ among programming languages, with variations in accuracy improvement percentages ranging from 2% to 19%. This study contributes to the ongoing discourse on developer expertise evaluation, highlighting the potential benefits of drawing from multiple data sources.

Keywords: *Developer Expertise Evaluation, Data Analysis, Machine Learning Algorithms, Serverless Functions, Software Development.*

1. Introduction

In the domain of software development, the ability to accurately evaluate developer expertise has become paramount [1-5]. This emphasis on expertise evaluation is not just a theoretical concern but has practical implications, especially in the evolving landscape of serverless functions. Serverless functions, often referred to as Function as a Service (FaaS) [6], simplify the development process by offloading infrastructure management to cloud providers. Recent data suggests that over 40% of companies have integrated serverless functions into their workflows, drawn by their scalability, cost-effectiveness, and the convenience of reduced infrastructure management [7][8].

With the increasing adoption of serverless functions, there's a growing need to understand the expertise behind the functions being developed. Developers frequently integrate functions developed by others into their projects. In such contexts, assessing the expertise of the original developer is crucial to ensure the reliability and efficiency of the integrated functions. Evaluating developer expertise in serverless functions is particularly important as it can significantly impact the quality and performance of the applications that use these functions.

Furthermore, as serverless functions are often developed using "Target Languages" like Java, Python, NodeJs[3], Ruby, Go, and C#—selected for their compatibility with serverless architectures and their widespread use [9] it becomes imperative to evaluate expertise specifically within these languages to ensure the quality and efficiency of serverless applications.

The accurate assessment of developer expertise in the broader software development field has been the focus of numerous investigations [1-4], [10–14] . While many works have been conducted in this area, several challenges persist. For instance, some studies have found that simple metrics, such as counting commits, might not be a reliable indicator of expertise within specific libraries or frameworks. Others have introduced tools that leverage Natural Language Processing (NLP) to pinpoint expertise, but these often rely solely on data from a single platform like GitHub. There's also a recognized need to merge data

[*] Manuscript received: 2023 September 15, Revised, 2024 February 14, Accepted, 2024 March 18.

[1] Graduate Student, Department of Computer Engineering, Ferdowsi University of Mashhad, Iran.

[2] Corresponding author. Associate Professor, Department of Computer Engineering, Ferdowsi University of Mashhad, Iran. **Email**: rasoolzadegan@um.ac.ir

[3] While NodeJs technically serves as a runtime environment facilitating the execution of JavaScript code on the server side, it is often colloquially referred to as a programming language due to its prevalent standalone usage in discourse. In this paper, we refer to it as one of our target languages

from multiple platforms, such as GitHub and Stack Overflow, but many existing approaches primarily focus on user profiles and specific APIs.

Building on these valuable insights from prior research, our study endeavors to bridge the identified gaps. Specifically, we aim to offer a fresh perspective on developer expertise by tapping into both GitHub and Stack Overflow, thereby opening a broader window of feature collecting, especially in the context of serverless functions. We've observed that while some research has adeptly employed machine learning classifiers, a predominant reliance on a single data platform suggests an opportunity for enhancement. Our work underscores the significance of adopting a multi-platform approach, which we believe can pave the way for a more holistic understanding of developer expertise.

Evaluating developer expertise, especially within serverless functions, requires a systematic approach. Following this notion, our research utilized the official GitHub REST API[4] for data extraction. We initially identified 408 GitHub repositories related to serverless functions. From these, we selected the top 150 to ensure representation across different target languages. On GitHub, we extracted 13 activity-related features, providing insights into a contributor's activities and expertise. Our interpretation of these features was informed by Montandon's work [11].

Turning to Stack Overflow, we sought to link contributors to their Stack Overflow profiles using the official StackAPI[5]. This allowed us to extract 9 additional features related to their activity on this platform. After data collection, we invited contributors to self-assess their expertise on a 0 to 5 scale. Of the 2539 emails we sent, 237 were answered, leading to a response rate of about 9.3%. This feedback aided in initially labeling our dataset.

After data extraction, we engaged in preprocessing to manage data-related challenges. For analysis, we employed machine learning algorithms like SVM [15], [16], Random Forest [17], Gradient Boosting [18], and Logistic Regression [19]. Initial results showed a preference for SVM and Random Forest in several datasets. When compared to other research, our findings suggested potential benefits from using data from both Stack Overflow and GitHub. The efficacy varied by target language: NodeJs exhibited an accuracy increase of approximately 19%, while C# showed an increase of about 2%, resulting in an average improvement of around 10.7%. Having outlined our proposed method, we sought to address these two specific research questions:

RQ1. Which machine learning algorithms are most effective in evaluating developer expertise in serverless functions based on the extracted features? The answers to this question, based on our comparative analyses of different algorithms, are discussed in the 'Evaluation' section.

RQ2. What features or metrics are most indicative of a developer's expertise in serverless functions? The insights related to this question, derived from our feature importance analysis, can be found in the 'Evaluation' section, specifically in the third part of that section.

In our research, we've made several contributions to evaluating developer expertise in serverless functions. Notably, we've adopted a dual-platform data extraction approach, gathering activity-based features of contributors[6] from both GitHub and Stack Overflow. This approach aims to provide a more detailed perspective on a developer's engagement by leveraging data from two major platforms. From this extraction, we've compiled six language-specific datasets, representing developer activities in serverless functions for the respective target languages. Importantly, by making these datasets publicly available, we aim to foster collaborative research and encourage further exploration in this domain. This detailed approach allows for a nuanced evaluation based on the specific programming language. Additionally, we've conducted a feature importance analysis using SHAP values to understand the relative importance of each extracted feature. This step helps in discerning which activities might be more indicative of a developer's expertise in serverless functions.

Our proposed method caters not only to serverless functions but also have versatility for broader software development contexts. This adaptability accentuates the potential of our techniques in navigating the multifaceted challenges of contemporary software development. Furthermore, our research underscores the imperative of a holistic, multi-platform approach in developer expertise evaluation, with implications for refining recruitment strategies in the industry and fostering a platform proficiency within academic frameworks.

The remainder of this paper is structured as follows: The next section delves into the Background, providing a foundational understanding of the domain and contextualizing our work within existing literature. Following this, we present our Proposed Method, detailing the approach and techniques we employed. The Evaluation section then discusses our findings, shedding light on the efficacy and implications of our method. We subsequently address potential Threats to Validity, ensuring a transparent and critical discussion of our study's limitations. The paper concludes with a Conclusion section, summarizing our key contributions, and then looks ahead to Future Directions, suggesting potential avenues for further research and exploration in this

[4] https://docs.github.com/en/rest?apiVersion=2022-11-28

[5] https://stackapi.readthedocs.io/en/latest

[6] In this study, term 'contributors' refers to developers active in serverless functions.

domain.

2. Background

The rapid evolution of software development has led to the emergence of various platforms where developers collaborate, share knowledge, and showcase their expertise. Platforms like GitHub and Stack Overflow have become central to this ecosystem, providing a wealth of data that can be mined to understand developer expertise and behavior. Several studies have delved into this realm, each offering unique insights and methodologies [5].

Vasilescu et al. embarked on a comprehensive exploration of the intricate relationship between Stack Overflow and GitHub activities, Their exploration spanned three distinct levels: macro, intermediate, and micro. Their macro-level analysis aimed to discern the differences between GitHub contributors based on their Stack Overflow involvement, probing whether activity on one platform could serve as a proxy for the other. The intermediate level delved into the distribution of developers' time between GitHub commits and their Q&A activity on Stack Overflow. At the micro level, the temporal coordination between GitHub commits and Stack Overflow Q&A activities was scrutinized. This multifaceted analysis underscores the intertwined nature of developer activities across these platforms, emphasizing the potential influence of participation on one platform over the other [12].

In a similar vein, Song et al. sought to profile developer expertise by harnessing data from both Stack Overflow and GitHub. The research underscored the challenges of profiling expertise based solely on a single platform, given the sparsity of expertise matrices for both Stack Overflow and GitHub. By integrating data from both platforms, the study illuminated the multifaceted nature of developer expertise, underscoring the potential benefits of a cross-community approach. Such a collaboration-aware method can potentially mitigate challenges like unanswered questions on Stack Overflow and delayed responses to pull requests on GitHub [2].

The realm of developer expertise assessment witnessed a novel approach with the introduction of "CVExplorer," a tool designed to identify potential developer candidates by meticulously analyzing their contributions to open-source projects on GitHub. By mining skills from GitHub contributions, the tool offers recruiters a more accurate representation of a developer's skills, emphasizing the importance of real-world contributions in assessing developer expertise. This approach, which transcends the traditional reliance on self-authored CVs, underscores the evolving paradigms in developer assessment and recruitment [14].

Building on this Constantinou and Kapitsaki delved deep into the nuances of developer expertise and their roles in software technologies. Their research introduced the concept of "core expertise," signifying the primary domain or technology group where a developer is most prolific. By employing various metrics and data from platforms like Stack Overflow and GitHub, the study provided a granular analysis of how developers transition between roles and how their expertise can be categorized. Such insights are pivotal in understanding the evolutionary trajectory of developers and the multifarious roles they assume over time [3].

Tian et al. introduced a novel approach aimed at constructing a cross-platform expert recommendation system by synergizing datasets from GitHub and Stack Overflow. This system, designed to spotlight top expert developers, or "geek talents," underscores the value of expert recommendation systems in the open-source community and for companies at large. By leveraging various attributes of user profiles, platform-specific APIs, and multiple account matching strategies, the system can adeptly identify top experts in specific technology fields. Such a method offers a fresh perspective on recommending top expert developers, emphasizing the increasing importance of these platforms in the software development community [4]. Santos et al. ventured into the domain of mining software repositories with the primary objective of identifying library experts. By analyzing the source code of collaborative projects on GitHub, the study introduced a method that ranks developers based on five dimensions of skills. Preliminary results from this research underscored the method's capability of identifying relevant users of specific libraries, emphasizing the importance of GitHub as a platform to showcase developers' knowledge and skills. Such a structured approach offers a comprehensive evaluation of a developer's proficiency, highlighting the potential benefits for recruitment and human resource allocation [13].

Oliveira et al. embarked on a comprehensive empirical study to identify library experts by analyzing source code. By evaluating the strategy with popular Java libraries and conducting an online survey with developers, the study provided insights into the challenges and methodologies of identifying library experts based on code analysis. The findings underscored that traditional metrics like "Lines of Code" or "Number of Commits" might not be sufficient indicators of a developer's expertise with specific libraries. Such insights are pivotal in understanding the nuances of developer expertise in specific libraries and the potential limitations of certain metrics in the identification process [10].

Lastly, Montandon's research focused on identifying experts in popular JavaScript libraries by mining GitHub data. By integrating repository mining with developer surveys, Montandon provided a robust method for pinpointing expertise in specific software libraries and frameworks. While this approach emphasized the importance of combining multiple data sources for accurate assessments, our research Recognizing the potential limitations in Montandon's approach, we advocate for a broader spectrum of features, underscoring the significance of a more detailed and comprehensive feature set. This expanded perspective is particularly crucial in the realm of serverless functions, where the landscape is rapidly evolving and the nuances of developer expertise are multifaceted [11]. In light of the existing studies, our research aims to enhance the understanding of developer expertise by amalgamating data from both GitHub and Stack Overflow. While we draw inspiration from the works mentioned, our unique contribution lies in showcasing the importance of adding more detailed features, offering a richer and more holistic understanding of developer expertise.

3. Proposed Method

We employed a structured research method to assess contributors' expertise in a balanced manner, drawing from their activities on GitHub and Stack Overflow. The method was divided into three main phases: Data Collection, Data Analysis, and Model Training.

Phase 1. Data Collection

In this phase, we aimed to gather data from GitHub and Stack Overflow to understand contributors' activities. Using the REST APIs of both platforms, we followed a structured process that began with the selection of repositories and culminated in the formation of language-specific datasets. This method was designed to ensure that our data was up-to-date and relevant to our research objectives. The steps outlined in Figure 1 provide a detailed breakdown of our data collection approach.

Step 1. GitHub Profile Collection for Contributors

We initially amassed a collection of 408 repositories pertinent to serverless functions. To optimize the scope of our study and ensure manageability, these repositories were filtered based on their number of stars. This criterion narrowed down our dataset to 150 repositories, striking a balance between comprehensiveness and feasibility. After finalizing the repository list, we retrieved the contributors associated with each repository. For each contributor, we began by extracting vital details such as email, display name, location, and more to establish a foundational profile. We then conducted a deep dive into the user's GitHub activities, encompassing metrics like the number of commits, code churn, and import statements. This provided a detailed picture of the contributor's engagement and coding habits. Moreover, for our GitHub data extraction, we discerned 13 activity-related features, each offering a perspective into a contributor's engagement and expertise. It's noteworthy to mention that our characterization and terminology for these features have been predominantly informed by Montandon's work [11], which stands as a cornerstone in our research approach. For instance, as elaborated in Table 1, "Client Projects" denotes repositories encompassing code in any of our target languages, thus serving as a metric to assess developers' acumen in these specific languages. Conversely, "Client Files" zooms in to highlight files within these repositories written in the target languages, offering a more granular view of the contributor's expertise.

Having collected the GitHub features, we then turned our attention to extracting features from Stack Overflow.

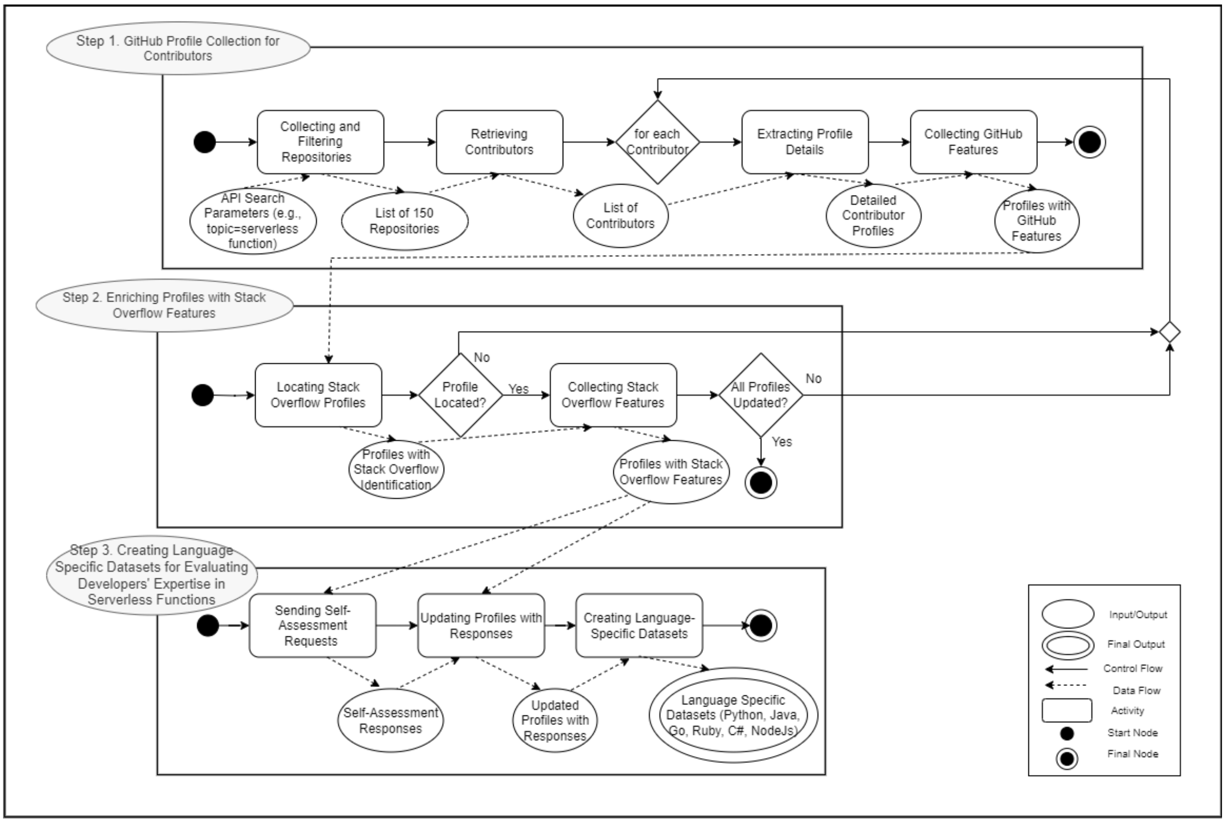
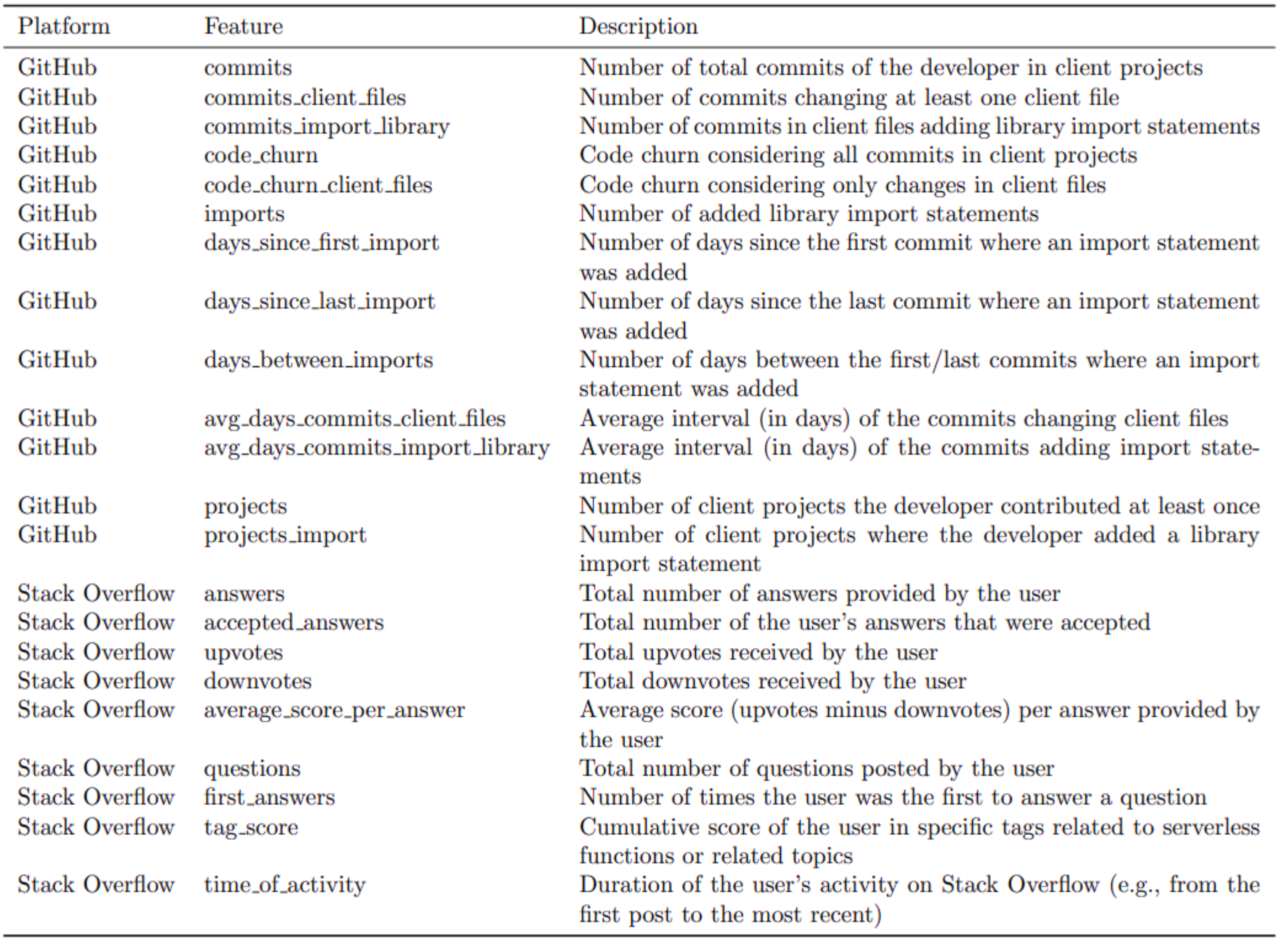

Step 1. GitHub Profile Collection for Contributors
Collecting and Filtering Repositories
Retrieving Contributors
for each Contributor
Extracting Profile Details
Collecting GitHub Features
API Search Parameters (e.g., topic=serverless function)
List of 150 Repositories
List of Contributors
Detailed Contributor Profiles
Profiles with GitHub Features
Step 2. Enriching Profiles with Stack Overflow Features
Locating Stack Overflow Profiles
Profile Located?
No
Yes
Collecting Stack Overflow Features
All Profiles Updated?
No
Yes
Profiles with Stack Overflow identification
Profiles with Stack Overflow Features
Step 3. Creating Language Specific Datasets for Evaluating Developers' Expertise in Serverless Functions
Sending Self-Assessment Requests
Updating Profiles with Responses
Creating Language-Specific Datasets
Self-Assessment Responses
Updated Profiles with Responses
Language Specific Datasets (Python, Java, Go, Ruby, C#, NodeJs)
Input/Output
Final Output
Control Flow
Data Flow
Activity
Start Node
Final Node

Figure 1. Process of the Data Collection Phase in the Proposed Method

Table 1 Extracted Features from GitHub and Stack Overflow



Platform	Feature	Description
GitHub	commits	Number of total commits of the developer in client projects
GitHub	commits_client_files	Number of commits changing at least one client file
GitHub	commits_import_library	Number of commits in client files adding library import statements
GitHub	code_churn	Code churn considering all commits in client projects
GitHub	code_churn_client_files	Code churn considering only changes in client files
GitHub	imports	Number of added library import statements
GitHub	days_since_first_import	Number of days since the first commit where an import statement was added
GitHub	days_since_last_import	Number of days since the last commit where an import statement was added
GitHub	days_between_imports	Number of days between the first/last commits where an import statement was added
GitHub	avg_days_commits_client_files	Average interval (in days) of the commits changing client files
GitHub	avg_days_commits_import_library	Average interval (in days) of the commits adding import statements
GitHub	projects	Number of client projects the developer contributed at least once
GitHub	projects_import	Number of client projects where the developer added a library import statement
Stack Overflow	answers	Total number of answers provided by the user
Stack Overflow	accepted_answers	Total number of the user's answers that were accepted
Stack Overflow	upvotes	Total upvotes received by the user
Stack Overflow	downvotes	Total downvotes received by the user
Stack Overflow	average_score_per_answer	Average score (upvotes minus downvotes) per answer provided by the user
Stack Overflow	questions	Total number of questions posted by the user
Stack Overflow	first_answers	Number of times the user was the first to answer a question
Stack Overflow	tag_score	Cumulative score of the user in specific tags related to serverless functions or related topics
Stack Overflow	time_of_activity	Duration of the user's activity on Stack Overflow (e.g., from the first post to the most recent)

Step 2. Enriching Profiles with Stack Overflow Features

To augment our dataset, we aimed to identify the Stack Overflow profile corresponding to each GitHub contributor. We began by initiating a search using the contributor's GitHub username. If any list of Stack Overflow profiles was returned, we took additional steps to ensure the authenticity and relevance of the identified profile. Specifically, we applied a method based on probabilistic record linkage [32], a well-established technique in data integration. This method involves calculating a similarity score based on the probabilities of agreement and disagreement for each attribute (website URL, location, bio, and company affiliation). These probabilities were computed based on the distribution of values in each field in our dataset. The profile with the highest similarity score was selected for further analysis. In situations where no profiles met our stringent criteria, we opted for the first profile returned by the search results, given StackAPI's tendency to list the most relevant user first.

To complement our GitHub data, we delved deeply into Stack Overflow, aiming to extract features that would provide a detailed view of each contributor's expertise and engagement. Starting with the Stack Overflow profile of each contributor, identified in the previous step, we centered our efforts on the target languages. For each language, we:

(1) Retrieved answers and questions provided by the user, which not only showcased their expertise but also their level of engagement with the community.

(2) Computed various metrics that reflected the user's overall activity and reputation for that target language (Tag Score) on Stack Overflow. This encompassed the number of answers they've given, upvotes received, downvotes given, and other related metrics. find their detailed representation in Table 1

(3) Consolidated these features and stored them in a CSV file, ensuring they were organized and ready for subsequent analysis.

Our research method heavily relied on the REST APIs of both GitHub and Stack Overflow. By integrating the GitHub REST API with the PyGitHub Python library, we ensured timely and accurate data extraction from GitHub. Similarly, our use of StackAPI streamlined our interactions with Stack Overflow, particularly in retrieving user profiles and associated data. We opted for these APIs over pre-existing datasets to ensure we were working with the most up-to-date online data, reinforcing the repeatability of our method. Given the dynamic nature of expertise, our method was designed to be adaptable, allowing for potential replication in future studies. A crucial part of our data collection process was managing the rate limits imposed by both platforms. To ensure a smooth data collection process, we sequentially extracted data, first focusing on GitHub features for each contributor and then moving to Stack Overflow.

Step 3. Creating Language Specific Datasets for Evaluating Developers' Expertise in Serverless Functions *We initiated this phase by reaching out to contributors for a self-assessment of their expertise. Each contributor received an email detailing our research objectives and was asked to rate their expertise on a scale from 0 (no expertise) to 5 (expert level). Supported by several studies as a reliable measure of expertise [11], [23], [24], this method yielded a response rate of approximately 9.33% from the 2539 emails dispatched.*

After updating the contributors' profiles with their self-assessed scores, we curated separate datasets for each of the target languages: Java, Python, NodeJs, Ruby, Go, and C#. These datasets encapsulated the contributors' activities specific to the respective programming language, with their self-assessed scores serving as the labels. This approach ensured a granular understanding of developer expertise in serverless functions, tailored to the nuances of each language.

Phase 2. Data Analysis

Upon completion of the data collection phase, we obtained six distinct datasets. Each dataset encapsulates 22 features detailing developers' activities on both GitHub and Stack Overflow, specific to each of our target languages. In the next step, we transitioned to the data analysis phase, which is further divided into two main steps:

Step 1- Data Visualization

Our initial exploration began with visualizing the distribution of developer expertise across our target languages, as depicted in Figure 2. For clarity in the figure, we've categorized our binned labels as follows: ratings of 0 and 1 are denoted as "Novice (0)", ratings of 2 and 3 are labeled "Intermediate (1)", and ratings of 4 and 5 are classified as "Expert (2)". We will delve into the binning process in a subsequent section. The visualization reveals compelling insights. For instance, languages like Python and Ruby, which are often considered beginner-friendly, had a significant portion of their developer base in the novice category.

This suggests that these languages are more accessible for beginners. In contrast, languages such as Java, a prevalent choice in enterprise settings, showcased a more balanced expertise distribution, indicating a diverse user base with varying levels of expertise.

Moving forward, we examined the skewness across different programming languages, visualized in Figure 3. Skewness, a measure indicating the asymmetry of data distribution around the mean, revealed specific patterns. A notable observation was the positive skewness values for commits in languages like Go, NodeJs, and Ruby. This suggests a scenario where a majority of users have fewer commits, but a select few exhibit exceptionally high commit counts. This skewness indicates that while most developers contribute at a moderate pace, there are a few highly active developers who contribute significantly more. Understanding this skewness is pivotal as it influences our data interpretation and subsequent modeling strategies [20]. Our analysis subsequently delved into correlation. It is crucial to emphasize that correlation does not imply causation [21]. Particularly in the realm of expertise, it is common for features to exhibit high correlation. However, even if two variables appear to move in tandem, this does not necessarily indicate a cause-and-effect relationship. For example, we observed a high correlation between the number of commits and the number of pull requests made by a developer. While these two variables are correlated, it does not necessarily mean that making more commits causes a developer to make more pull requests, or vice versa. It could simply be that more active developers tend to both commit and pull request more frequently.

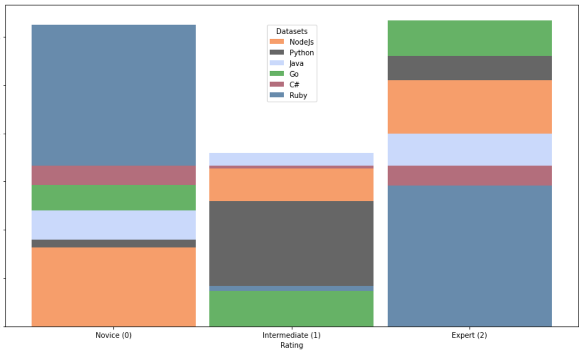
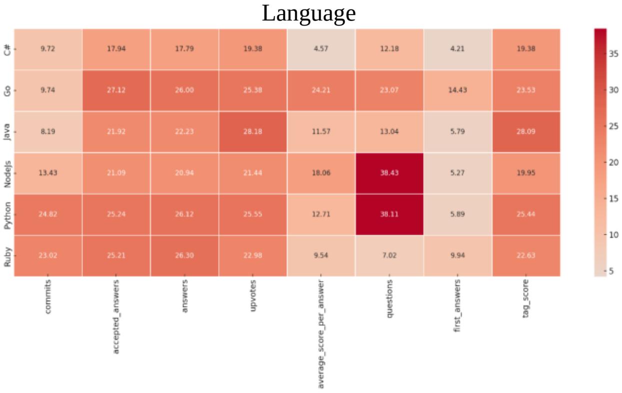


Figure 2. Incremental Distribution of Ratings Across Target Language



	commits	accepted_answers	answers	upvotes	average_score_per_answer	questions	first_answers	tag_score
C#	9.72	17.94	17.79	19.38	4.57	12.18	4.21	19.38
Go	9.74	27.12	26.00	25.38	24.21	23.07	14.43	23.53
Java	8.19	21.92	22.23	28.18	11.57	13.04	5.79	28.09
NodeJs	13.43	21.09	20.94	21.44	18.06	38.43	5.27	19.95
Python	24.82	25.24	26.12	25.55	12.71	38.11	5.89	25.44
Ruby	23.02	25.21	26.30	22.98	9.54	7.02	9.94	22.63

Figure 3. Skewness of Features Across Target Languages

Step 2- Data Preprocessing

After the data analysis phase, we transitioned to data preprocessing, a crucial step to ensure the data was primed for modeling. This phase addressed several challenges, including missing values, skewness, feature selection, label binning, and label imputation.

Handling Missing Values: In addressing missing values, we observed that their presence in our dataset often signified a developer's inactivity for a specific feature. Given this observation, we logically imputed these missing values with zero, symbolizing no activity.

Addressing Skewness: Skewness presented another challenge. Some features in our dataset displayed right-skewed distributions. To make the data more suitable for modeling, we applied a logarithmic transformation [22] to these skewed features, stabilizing their variance and approximating a normal distribution. However, before this transformation, we had to manage zeros in the data, as the logarithm of zero is undefined. To counteract this, we added a constant of 1 to the respective columns.

Feature Selection: Feature selection emerged as a pivotal aspect of our preprocessing [23]. Ensuring that our model is trained on relevant features is paramount for enhancing its performance and interpretability. Our multifaceted approach to feature selection began with an examination of correlations. While correlation provides a measure of the linear relationship between two variables, it doesn't capture non-linear relationships or causation, making sole reliance on it potentially misleading [21]. We also incorporated domain knowledge for the feature selection, by understanding the context of each feature in the software development realm, we made informed decisions. For example, while the number of answers and accepted answers on Stack Overflow might be correlated, the emphasis on quality in the latter could offer more valuable insights. Additionally, we employed the Gradient Boosting technique, an ensemble learning method, to gain insights into feature importance. This technique ranked features based on their significance in influencing the model's predictions. Armed with these analyses, we made informed decisions on which features to discard, ensuring our model was built on a relevant feature set, as visualized in figure 4.

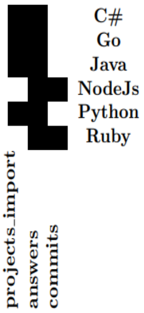


Figure 4: Dropped Features for Each Programming Language

Label Binning: Following feature selection, we addressed the challenge of label binning. Our dataset contained ratings ranging from 0 to 5, indicative of the expertise level. To streamline our modeling process and make the problem more approachable, we categorized these ratings into three broader categories. Ratings between 0 and 1 were represented as Novice (0), those between 2 and 3 as Intermediate (1), and ratings between 4 and 5 were categorized as Expert (2). This categorization ensured our models could predict expertise levels in broader, more generalizable categories, enhancing the interpretability and applicability of the results.

Label Imputation: Transitioning from the binning process, we confronted another intricate challenge: Label Imputation. In the vast landscape of data-driven research, addressing the absence of labels is a multifaceted endeavor. To adeptly navigate this, we adopted a dual-strategy approach. This method seamlessly integrated the domain-centric heuristic labeling [24] with the algorithmic precision of k-Nearest Neighbors (kNN) [25] imputation. The rationale behind employing heuristic labeling as our initial step stemmed from the nature of our data. Given that some of our labels represented minority classes, relying solely on kNN label imputation could introduce biases towards the majority classes. Heuristic labeling, while dependent on the accuracy of the criteria set for label determination, offers a way to balance the label distribution without such biases. However, recognizing the limitations of heuristic accuracy, we used this method to label only 10-12% of the dataset. This preliminary step aimed to create a more balanced label distribution, setting the stage for the subsequent kNN imputation. The combination of heuristic labeling and kNN imputation allowed us to leverage the strengths of both methods, providing a balance between domain-centric insights and algorithmic precision, and mitigating the potential weaknesses of each method when used alone.

*1: **Heuristic Labeling:*** Heuristic labeling leverages domain insights and data patterns to make informed decisions about labels. This method, while not exhaustive, offers a valuable starting point, especially when dealing with imbalanced datasets. Our heuristic algorithm works iteratively. By adjusting criteria based on the quantiles of key features, we assign scores and labels to each instance. The process either converges to a desired label distribution or stops after a predetermined number of iterations.

***2: k-Nearest Neighbors (kNN) Imputation**:* In advancing our research, we transitioned from heuristic labeling to the more nuanced kNN imputation. The underlying principle of the kNN algorithm is rooted in similarity; for any unlabeled data point, it identifies its ‘k’ closest labeled neighbors and adopts the predominant label among them. The selection of ‘k’ is crucial, and through meticulous testing, we ascertained the ideal ‘k’ for distinct data subsets. In our case, we performed an iterative evaluation to determine the optimal ‘k’ value and found that k=5 yielded the best results. Following the primary imputation, certain labels emerged as fractional. To ensure consistency with our discrete categories—Novice, Intermediate, and Expert—we adjusted these by rounding to the closest whole number. It’s worth noting that we used kNN in conjunction with heuristic labeling for the label binning task. While our two-phase approach aimed to provide the most accurate labels possible, it’s worth noting that any label imputation method can introduce errors. For instance, heuristic labeling, while effective in balancing the dataset, might not always capture the nuances of individual developer expertise. Similarly, kNN, though a more common method, can sometimes be influenced by noisy neighbors, leading to potential mislabeling.

After implementing our dual-strategy, the datasets displayed more balanced distributions across various programming languages. For example, widely-used languages like Java showed a more uniform distribution, whereas specialized languages like Go had a higher concentration of expert developers. Notably, after the labeling process, there was a noticeable increase in the number of novice developers. This trend mirrors the real-world scenario where many individuals begin coding, but only a select few achieve expert status.

Upon addressing missing labels with heuristic labeling and kNN imputation, we tackled class imbalance using the Synthetic Minority Over-sampling Technique (SMOTE) [26], [27]. For each dataset, we set target percentages for the 'expert' class, like 15% for C# and 10% for others. We then split the data, ensuring a representative class distribution. Before applying SMOTE, we dynamically set the number of nearest neighbors based on the 'expert' instances in the training data. Using SMOTE, we balanced our datasets, enhancing model accuracy. However, SMOTE can introduce noise, potentially leading to model over-generalization on real-world data.

Phase 3. Model Training:

With our datasets now complete and labeled, we transitioned to the third phase of our research method, we directed our attention towards the pivotal aspect of model training. This phase is instrumental in elucidating the predictive capabilities inherent within our rigorously assembled datasets.

Given our objective of predicting developer expertise levels, we identified it as a multi-class classification problem. We experimented with various machine learning algorithms, including Random Forest (RF), Gradient Boosting (GB), Support Vector Machines (SVM), and Logistic Regression (LR), to assess their performance on our datasets. The rationale behind selecting these classifiers is multi-fold:

Random Forest and Gradient Boosting are both ensemble learning methods known for their high accuracy and ability to handle large datasets with higher dimensionality. They can effectively manage missing values and provide a good indicator of feature importance. Support Vector Machines are renowned for their power in high-dimensional spaces, which is particularly beneficial given the number of features in our dataset. They offer robustness, especially when the number of dimensions exceeds the number of samples. Logistic Regression while is a simpler algorithm, is effective for binary and multi-class classification problems. Its ease of implementation and interpretability make it a valuable tool in our arsenal [15], [17], [18].

In our study, we examined our six distinct datasets, each corresponding to a target language, alongside three datasets from the base article [11]. To understand the potential influence of incorporating Stack Overflow features, we employed a two-pronged analytical approach.

GitHub-Only Features: Initially, all Stack Overflow features were dropped from our datasets, leaving only the GitHub features. This step was inspired by our base article, which solely considered GitHub features. It is essential to note that the datasets derived from the base article [11] were accessible to the public. To maintain consistency and ensure an equitable comparison, we subjected these datasets to analogous preprocessing steps as those implemented for our own datasets. This approach not only aligned our method but also enhanced the performance of the models on the base article datasets. The results presented in Table 3 for the three base datasets surpass the performance metrics reported in their original paper[11], offering a more robust comparison.

Incorporating Stack Overflow Features: Subsequently, we reintroduced the Stack Overflow features to our six datasets and retrained our models. The performance results post this addition are showcased in Table 4. For each of the above approaches, we embarked on a systematic method. First, we loaded the respective datasets, each tailored to a specific approach, be it GitHub-only or incorporating Stack Overflow features. Then, the data was split into training and testing sets using stratified sampling [28], ensuring that each set accurately represented the overall class distribution. Subsequently, we standardized the features using the StandardScaler [29], ensuring they were on the same scale, which is crucial for effective model training. Next, for each classifier, we defined a set of hyperparameters. To find the optimal parameters, we employed GridSearchCV, an exhaustive search method over the specified parameter values for an estimator. This method pinpointed the parameters of the estimator that yielded the best results on the left-out validation set, ensuring our models were primed for optimal performance.

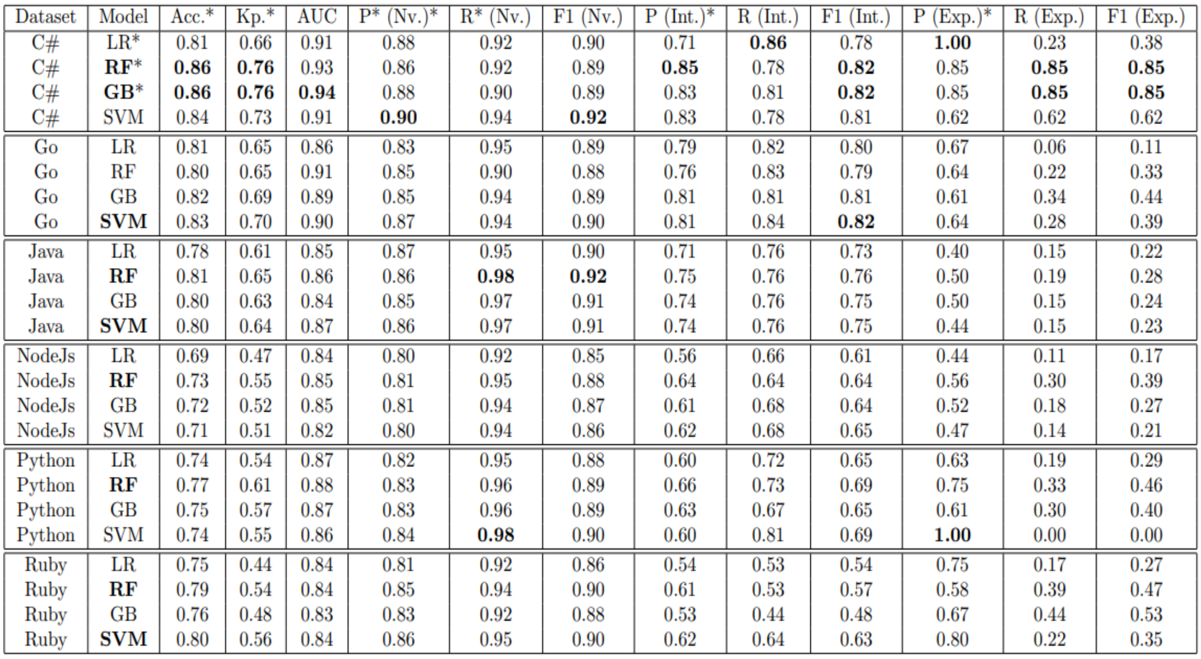
3.1 Assumptions and Limitations of the proposed method. In every research endeavor, certain assumptions guide the method, and inherent limitations bound the scope. This section elucidates the foundational assumptions underpinning our proposed method and highlights potential constraints that might influence the interpretation of our findings. Recognizing these factors ensures a nuanced understanding of our research outcomes.

Assumptions:

Data Completeness: We assumed that the data sourced from GitHub and Stack Overflow accurately represents the activities and contributions of developers. However, there might be private repositories or contributions that are not publicly accessible.

Feature Relevance: The features selected for model training were deemed relevant based on prior literature and domain knowledge. It's assumed that these features are indicative of a developer's expertise.

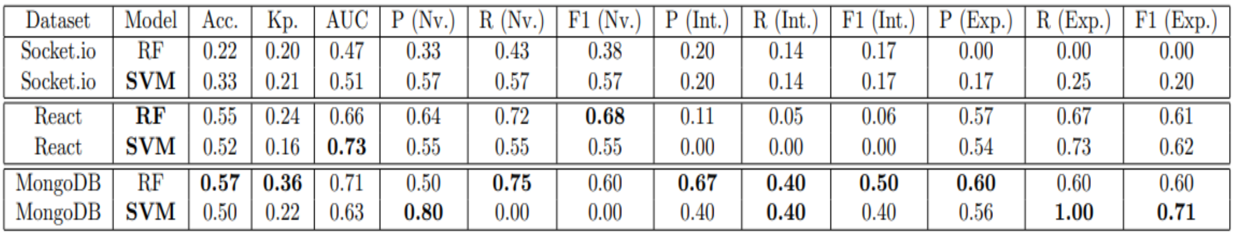
Table 2 Performance metrics of models trained using GitHub for our Target Languages Datasets



Dataset	Model	Acc.*	Kp.*	AUC	P* (Nv.)*	R* (Nv.)	F1 (Nv.)	P (Int.)*	R (Int.)	F1 (Int.)	P (Exp.)*	R (Exp.)	F1 (Exp.)
C#	LR*	0.81	0.66	0.91	0.88	0.92	0.90	0.71	**0.86**	0.78	**1.00**	0.23	0.38
C#	**RF***	**0.86**	**0.76**	0.93	0.86	0.92	0.89	**0.85**	0.78	**0.82**	0.85	**0.85**	**0.85**
C#	**GB***	**0.86**	**0.76**	**0.94**	0.88	0.90	0.89	0.83	0.81	**0.82**	0.85	**0.85**	**0.85**
C#	SVM	0.84	0.73	0.91	**0.90**	0.94	**0.92**	0.83	0.78	0.81	0.62	0.62	0.62
Go	LR	0.81	0.65	0.86	0.83	0.95	0.89	0.79	0.82	0.80	0.67	0.06	0.11
Go	RF	0.80	0.65	0.91	0.85	0.90	0.88	0.76	0.83	0.79	0.64	0.22	0.33
Go	GB	0.82	0.69	0.89	0.85	0.94	0.89	0.81	0.81	0.81	0.61	0.34	0.44
Go	**SVM**	0.83	0.70	0.90	0.87	0.94	0.90	0.81	0.84	**0.82**	0.64	0.28	0.39
Java	LR	0.78	0.61	0.85	0.87	0.95	0.90	0.71	0.76	0.73	0.40	0.15	0.22
Java	**RF**	0.81	0.65	0.86	0.86	**0.98**	**0.92**	0.75	0.76	0.76	0.50	0.19	0.28
Java	GB	0.80	0.63	0.84	0.85	0.97	0.91	0.74	0.76	0.75	0.50	0.15	0.24
Java	**SVM**	0.80	0.64	0.87	0.86	0.97	0.91	0.74	0.76	0.75	0.44	0.15	0.23
NodeJs	LR	0.69	0.47	0.84	0.80	0.92	0.85	0.56	0.66	0.61	0.44	0.11	0.17
NodeJs	**RF**	0.73	0.55	0.85	0.81	0.95	0.88	0.64	0.64	0.64	0.56	0.30	0.39
NodeJs	GB	0.72	0.52	0.85	0.81	0.94	0.87	0.61	0.68	0.64	0.52	0.18	0.27
NodeJs	SVM	0.71	0.51	0.82	0.80	0.94	0.86	0.62	0.68	0.65	0.47	0.14	0.21
Python	LR	0.74	0.54	0.87	0.82	0.95	0.88	0.60	0.72	0.65	0.63	0.19	0.29
Python	**RF**	0.77	0.61	0.88	0.83	0.96	0.89	0.66	0.73	0.69	0.75	0.33	0.46
Python	GB	0.75	0.57	0.87	0.83	0.96	0.89	0.63	0.67	0.65	0.61	0.30	0.40
Python	SVM	0.74	0.55	0.86	0.84	**0.98**	0.90	0.60	0.81	0.69	**1.00**	0.00	0.00
Ruby	LR	0.75	0.44	0.84	0.81	0.92	0.86	0.54	0.53	0.54	0.75	0.17	0.27
Ruby	**RF**	0.79	0.54	0.84	0.85	0.94	0.90	0.61	0.53	0.57	0.58	0.39	0.47
Ruby	GB	0.76	0.48	0.83	0.83	0.92	0.88	0.53	0.44	0.48	0.67	0.44	0.53
Ruby	**SVM**	0.80	0.56	0.84	0.86	0.95	0.90	0.62	0.64	0.63	0.80	0.22	0.35

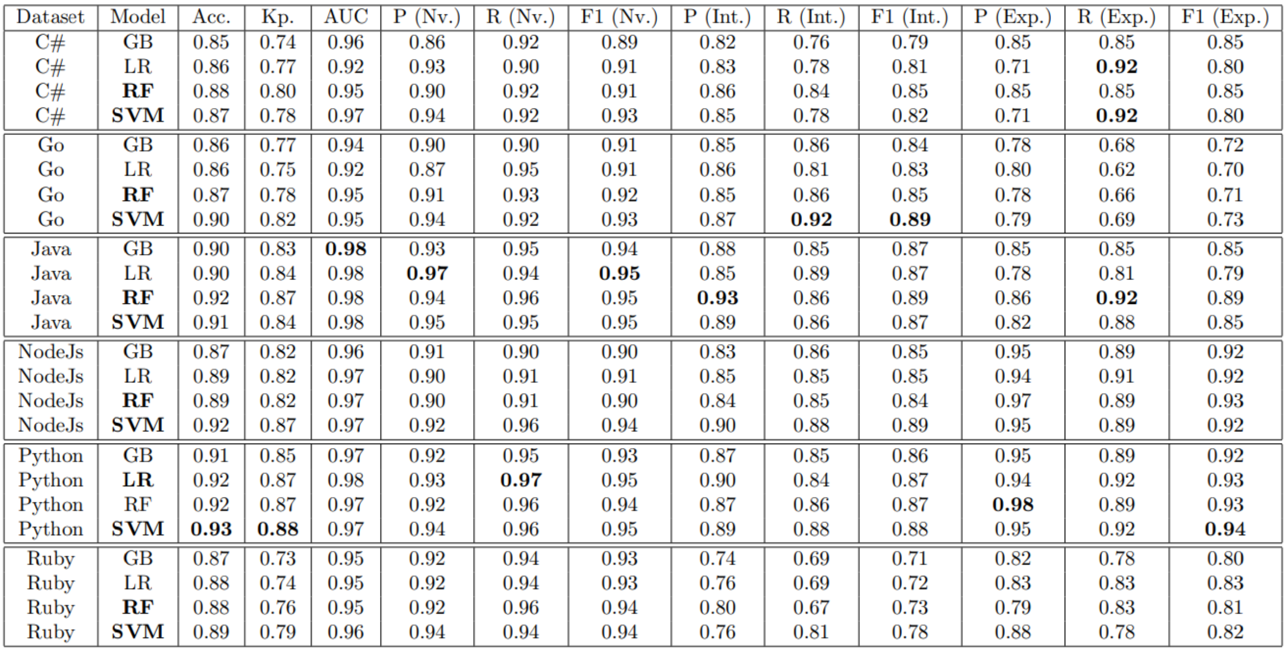
Key: Metrics - Acc. (Accuracy), Kp. (Kappa), AUC (Area Under the Curve), P (Precision), R (Recall), F1 (F1 Score); Expertise Levels - Nv. (Novice), Int. (Intermediate), Exp. (Expert); Models - LR. (Logistic Regression), RF. (Random Forest), GB. (Gradient Boosting).

Table 3 Performance metrics of models trained using GitHub for Datasets from the Base Article [11]



Dataset	Model	Acc.	Kp.	AUC	P (Nv.)	R (Nv.)	F1 (Nv.)	P (Int.)	R (Int.)	F1 (Int.)	P (Exp.)	R (Exp.)	F1 (Exp.)
Socket.io	RF	0.22	0.20	0.47	0.33	0.43	0.38	0.20	0.14	0.17	0.00	0.00	0.00
Socket.io	**SVM**	0.33	0.21	0.51	0.57	0.57	0.57	0.20	0.14	0.17	0.17	0.25	0.20
React	**RF**	0.55	0.24	0.66	0.64	0.72	**0.68**	0.11	0.05	0.06	0.57	0.67	0.61
React	**SVM**	0.52	0.16	**0.73**	0.55	0.55	0.55	0.00	0.00	0.00	0.54	0.73	0.62
MongoDB	RF	**0.57**	**0.36**	0.71	0.50	**0.75**	0.60	**0.67**	**0.40**	**0.50**	**0.60**	0.60	0.60
MongoDB	**SVM**	0.50	0.22	0.63	**0.80**	0.00	0.00	0.40	**0.40**	0.40	0.56	**1.00**	**0.71**

Table 4 Performance metrics of models trained using GitHub and Stack Overflow features



Dataset	Model	Acc.	Kp.	AUC	P (Nv.)	R (Nv.)	F1 (Nv.)	P (Int.)	R (Int.)	F1 (Int.)	P (Exp.)	R (Exp.)	F1 (Exp.)
C#	GB	0.85	0.74	0.96	0.86	0.92	0.89	0.82	0.76	0.79	0.85	0.85	0.85
C#	LR	0.86	0.77	0.92	0.93	0.90	0.91	0.83	0.78	0.81	0.71	**0.92**	0.80
C#	**RF**	0.88	0.80	0.95	0.90	0.92	0.91	0.86	0.84	0.85	0.85	0.85	0.85
C#	**SVM**	0.87	0.78	0.97	0.94	0.92	0.93	0.85	0.78	0.82	0.71	**0.92**	0.80
Go	GB	0.86	0.77	0.94	0.90	0.90	0.91	0.85	0.86	0.84	0.78	0.68	0.72
Go	LR	0.86	0.75	0.92	0.87	0.95	0.91	0.86	0.81	0.83	0.80	0.62	0.70
Go	**RF**	0.87	0.78	0.95	0.91	0.93	0.92	0.85	0.86	0.85	0.78	0.66	0.71
Go	**SVM**	0.90	0.82	0.95	0.94	0.92	0.93	0.87	**0.92**	**0.89**	0.79	0.69	0.73
Java	GB	0.90	0.83	**0.98**	0.93	0.95	0.94	0.88	0.85	0.87	0.85	0.85	0.85
Java	LR	0.90	0.84	0.98	**0.97**	0.94	**0.95**	0.85	0.89	0.87	0.78	0.81	0.79
Java	**RF**	0.92	0.87	0.98	0.94	0.96	0.95	**0.93**	0.86	0.89	0.86	**0.92**	0.89
Java	**SVM**	0.91	0.84	0.98	0.95	0.95	0.95	0.89	0.86	0.87	0.82	0.88	0.85
NodeJs	GB	0.87	0.82	0.96	0.91	0.90	0.90	0.83	0.86	0.85	0.95	0.89	0.92
NodeJs	LR	0.89	0.82	0.97	0.90	0.91	0.91	0.85	0.85	0.85	0.94	0.91	0.92
NodeJs	**RF**	0.89	0.82	0.97	0.90	0.91	0.90	0.84	0.85	0.84	0.97	0.89	0.93
NodeJs	**SVM**	0.92	0.87	0.97	0.92	0.96	0.94	0.90	0.88	0.89	0.95	0.89	0.92
Python	GB	0.91	0.85	0.97	0.92	0.95	0.93	0.87	0.85	0.86	0.95	0.89	0.92
Python	**LR**	0.92	0.87	0.98	0.93	**0.97**	0.95	0.90	0.84	0.87	0.94	0.92	0.93
Python	RF	0.92	0.87	0.97	0.92	0.96	0.94	0.87	0.86	0.87	**0.98**	0.89	0.93
Python	**SVM**	**0.93**	**0.88**	0.97	0.94	0.96	0.95	0.89	0.88	0.88	0.95	0.92	**0.94**
Ruby	GB	0.87	0.73	0.95	0.92	0.94	0.93	0.74	0.69	0.71	0.82	0.78	0.80
Ruby	LR	0.88	0.74	0.95	0.92	0.94	0.93	0.76	0.69	0.72	0.83	0.83	0.83
Ruby	**RF**	0.88	0.76	0.95	0.92	0.96	0.94	0.80	0.67	0.73	0.79	0.83	0.81
Ruby	**SVM**	0.89	0.79	0.96	0.94	0.94	0.94	0.76	0.81	0.78	0.88	0.78	0.82

Model Applicability: We assumed that the machine learning models chosen for this study are suitable for the type of data and the problem at hand. The performance of these models might vary with different datasets or contexts.

Developer Overlap between NodeJs and MongoDB: In the base article [11], MongoDB is represented as NODE-MONGODB, the official NodeJs driver for the MongoDB database server. While there's likely a significant overlap between MongoDB and NodeJs developers, it's crucial to acknowledge that not all MongoDB developers may be proficient in NodeJs, and vice versa. For the purpose of a more direct comparison between our datasets and those of the base article, we operate under the assumption that developers associated with MongoDB also have expertise in NodeJs. Self-Assessment Reliability: We operate under the assumption that the self-assessments provided by contributors are accurate and offer a reliable representation of their expertise levels.

Limitations:

Data Bias: Relying solely on GitHub and Stack Overflow might introduce a bias, as developers might be active on other platforms or might not be active online at all. This could lead to an incomplete representation of a developer's true expertise. GitHub and Stack Overflow are widely used platforms in the developer community, but they may not fully represent the broader developer community. For example, developers who primarily use other platforms or who do not participate in online communities may not be well-represented. Furthermore, the behaviors and activities on these platforms may not fully reflect a developer's offline activities or their activities on other platforms. One potential way to mitigate this bias, which we discuss in our conclusion, is to integrate data from a wider range of developer platforms in future research.

External Validity: While our models showed promising results in the context of serverless functions, their applicability to other technological domains needs further validation.

Feature Limitations: While we integrated features from GitHub and Stack Overflow, other platforms like LinkedIn or TopCoder might offer additional insights that could enhance the model's predictive power. We attempted to use the public LinkedIn REST API7 to complement our data collection with user activity data from LinkedIn. However, to the best of our knowledge, the LinkedIn public API is no longer accessible. Furthermore, while we considered implementing a web crawler to gather data from LinkedIn, we decided against it. Although it's not illegal to collect publicly accessible data from the web, such an approach would be against LinkedIn's terms and conditions.

Model Constraints: Every machine learning model has its inherent limitations. For instance, linear models might not capture non-linear relationships well, and tree-based models might overfit on sparse data [30].

4. Evaluation

In response to RQ1, which inquires about the most effective machine learning algorithms for evaluating developer expertise in serverless functions based on the extracted features, we conducted a rigorous evaluation of our trained models. Once the models were trained with the best parameters; they underwent rigorous evaluation on the test set. We computed a range of metrics, including accuracy, kappa score, and AUC, to evaluate their performance. Additionally, we calculated precision, recall, and F1-score for each class - Novice, Intermediate, and Expert.

This detailed evaluation ensured we captured the nuances of each model's predictive capabilities, providing a detailed understanding of their strengths and limitations. By adhering to this method for both approaches, we ensured a consistent evaluation, allowing for a fair comparison between the utility of GitHub-only features and the combined GitHub and Stack Overflow features. The results for both approaches can be found in Table 2, Table 3, and Table 4. Following the presentation of results, we delve into a deeper analysis of the datasets. Initially, we focus on datasets using only GitHub features, subsequently, we discuss the datasets enhanced with Stack Overflow features and the resultant impact on model outcomes. The evaluation concludes with a feature importance analysis using the SHAP method, offering insights into the pivotal role of each feature. In this structured discussion, we aim to present a thorough understanding of our findings and their broader implications.

Part 1. Discussion Datasets with only GitHub Features

In the evaluation of target languages' datasets presented in Table 2, several trends and patterns emerge. RF model consistently exhibits robust performance across multiple datasets, often securing the lead in accuracy, Kappa, and F1 scores. This superior performance can be attributed to RF's ensemble nature, which amalgamates the results of numerous decision trees, offering a more generalized and resilient model. On the other hand, the SVM model also demonstrates commendable performance, particularly in precision. However, certain anomalies, such as in the Python dataset for the Exp. category, reveal that while SVM can predict with high confidence, it might occasionally overlook specific classes, resulting in a diminished recall.

[7] https://developer.linkedin.com/product-catalog

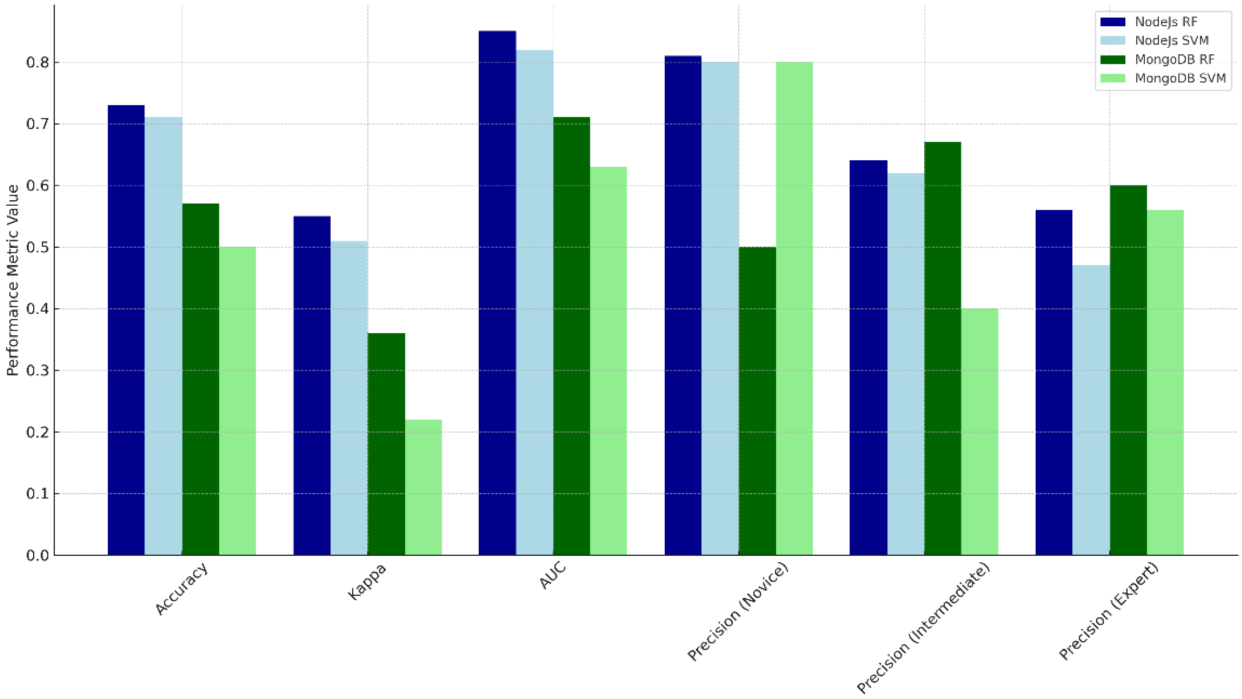


Figure 5. Comparison of performance metrics for NodeJs and MongoDB datasets

The GB model, another contender, frequently matches the performance of RF across datasets. GB's strength lies in its boosting algorithm, which zeroes in on challenging instances, potentially enhancing its performance on intricate datasets. Conversely, the LR model, inherently linear, occasionally falls short compared to tree-based or SVM models, especially when faced with datasets characterized by non-linear decision boundaries. Table 3, focusing on base work [11] results, highlights the challenges encountered in model predictions. The Socket.io dataset, for instance, presents notably low metrics for both RF and SVM models.

In Figure 5, we compared the performance of the Random Forest and SVM models for NodeJs and MongoDB datasets. The motivation behind selecting the NodeJs and MongoDB datasets for comparison is rooted in the anticipated overlap between their developer communities. Based on the premise that MongoDB, denoted as NODE-MONGODB — the official NodeJs driver for the MongoDB database server — would have a substantial confluence of developers skilled in both domains, these datasets were chosen for a deeper analysis. Upon analyzing the performance metrics, a distinct pattern becomes evident: models trained on the NodeJs dataset tend to surpass those trained on the MongoDB dataset across the majority of metrics. Moreover, within the NodeJs dataset, the RF model consistently outshines the SVM in parameters such as accuracy, kappa, and AUC. Both models demonstrate laudable precision in recognizing novice developers, underscoring a robust ability to accurately discern beginner-level expertise. Nonetheless, a discernible drop in precision is observed as we transition to higher levels of expertise, particularly in the SVM model for NodeJs. The performance metrics for MongoDB were influenced by multiple factors. A primary consideration is the label retrieval method, which predominantly hinged on self-assessment. While our datasets also utilized self-assessment, it is noteworthy that around 9% of our data was labeled based on genuine expertise levels. Our hybrid labeling approach, encompassing heuristic techniques and kNN label imputation, seemed to yield more consistent outcomes. This approach counteracts the inherent discrepancies often associated with sole reliance on self-assessments. The potential for developers to inaccurately evaluate their own skills introduces the risk of misclassification. Such disparities might be accentuated for MongoDB, suggesting potential variances between self-declared and actual proficiency. In addition, the constraints posed by a limited number of training instances cannot be overlooked. Despite our efforts in employing techniques like SMOTE to address dataset imbalance, the foundational issue of a restricted data sample might induce overfitting, consequently diminishing the model's generalization capabilities. For consistency, the same preprocessing steps were applied across all nine datasets, and it's worth mentioning that the performance results reported for the base work [11] witnessed an enhancement due to our preprocessing techniques.

Moreover, while accuracy is a widely used metric, its limitations become pronounced, especially in datasets with class imbalances. Other metrics like Kappa, AUC, precision, recall, and F1 score provide a more comprehensive view, capturing the model's performance nuances across various classes. In conclusion, while RF often stands out in performance metrics, it's crucial to consider each dataset's unique characteristics when choosing a model. The occasional unexpected or below-average metric emphasizes the importance of a holistic evaluation using a diverse set of metrics, ensuring a well-rounded understanding of model performance.

Part 2. Discussion on the Enhanced Datasets with Stack Overflow Features

Upon integrating Stack Overflow features into our datasets, as shown in Table 4, we observed a marked improvement in the performance metrics of the models.

This enhancement underscores the significance of feature engineering and the potential of external data sources in boosting model performance. For the C# dataset, the RF model stands out with the highest accuracy of 0.88 and a Kappa score of 0.80. SVM, with an AUC of 0.97, indicates its capability to distinguish between classes effectively. Interestingly, LR showcases a perfect recall of 0.92 for the Exp. category, emphasizing its strength in identifying true positives for this class. In the Go dataset, SVM takes the lead with an accuracy of 0.90 and a Kappa score of 0.82. Its performance in the Int. category, with a recall of 0.92 and an F1 score of 0.89, is noteworthy, suggesting its proficiency in classifying intermediate instances. For the Java dataset, RF demonstrates robust performance with an accuracy of 0.92 and a Kappa score of 0.87. Its precision of 0.93 for the Int. category is commendable. However, GB's AUC of 0.98 is the highest, indicating its superior ability to differentiate between the classes.

In the NodeJs dataset, SVM emerges as the top performer with an accuracy of 0.92 and a Kappa score of 0.87. Its F1 scores across all categories are consistently high, reflecting its balanced precision and recall.

For the Python dataset, SVM shines with the highest accuracy of 0.93 and a Kappa score of 0.88. Its performance in the Exp. category, with an F1 score of 0.94, is particularly impressive. LR's recall of 0.97 for the Nv. category is the highest, indicating its strength in identifying true positives for novice instances.

Lastly, in the Ruby dataset, SVM leads with an accuracy of 0.89 and a Kappa score of 0.79. Its performance in the Nv. category, with an F1 score of 0.94, is outstanding, suggesting its proficiency in classifying novice instances. The addition of Stack Overflow features has evidently bolstered the models' performance across the datasets. The enriched datasets provide a more detailed view of each instance, allowing the models to capture intricate patterns and relationships. It's worth noting that while we couldn't expand the base datasets to include Stack Overflow features due to privacy concerns related to email hashing, the similarity in nature between NODE-MONGODB (NodeJs driver for MongoDB) and NodeJs offers a reasonable point of comparison. Given the overlaps between MongoDB and NodeJs developers, this similarity can serve as a benchmark to assess the impact of the added features. We hypothesize that if we were able to expand the MongoDB dataset similarly, we would likely observe comparable improvements as seen with the NodeJs dataset.

In conclusion, the integration of external features, such as those from Stack Overflow, can substantially enhance model performance. The importance of feature engineering is evident, and the potential of utilizing external data sources in machine learning tasks is undeniable. Referring to the results depicted in Figure 6, it's clear that leveraging additional features leads to noticeable accuracy boosts across various programming languages. For example, the SVM model accuracy for the 'Go' language saw an increase from 83% using only GitHub features to 92% when integrated with Stack Overflow features, and the continuous pursuit of integrating relevant external data to achieve optimal model performance. We observed similar significant improvements for languages like 'Java', 'NodeJs', and 'Python'. However, for languages like 'C#' and 'Ruby', the enhancements were more modest. These findings underscore the value of broadening the feature space.

Part 3. Feature Importance Analysis

In response to RQ2, which seeks to identify the most indicative features or metrics of a developer's expertise in serverless functions, we employed the SHAP (SHapley Additive exPlanations) method. SHAP values offer a unified measure of feature importance, assigning each feature an importance value for a specific prediction. This method excels in providing both global interpretability—indicating the importance of each feature across the entire dataset—and local interpretability, which explains individual predictions. As illustrated in Figure 7 (a SHAP summary plot), the SHAP analysis offers a comprehensive perspective on the significance of each feature [31].

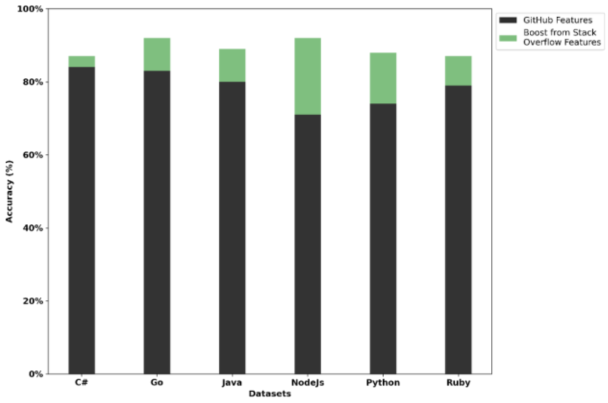


Figure 6: Comparison of SVM model accuracies using only GitHub features versus the improvement achieved by adding Stack Overflow features for target languages

Our analysis reveals the pivotal role of various features in predicting a developer's proficiency. The 'avg_days_commits_import_library' feature, denoting the average number of days between commits that import libraries, stands out as paramount. This metric suggests that a developer integrating new libraries frequently might be inclined towards proactive experimentation and learning. Similarly, the 'commits_import_library' feature, which reflects the number of such commits, can hint at the intricacy of applications a developer crafts. The 'time_of_activity' metric, capturing the span of a developer's activity, indicates sustained technological interest, which is crucial for continuous learning and expertise development.

Yet, insights aren't solely derived from GitHub activity. Additionally, 'commits_client_files', which counts commits altering at least one client file, sheds light on a developer's active involvement and contributions in a project's primary language. This could be interpreted as a sign of a developer's commitment to a project and their expertise in the project's main language. Integration of Stack Overflow features enriches our understanding. For instance, 'upvotes', ranking as the third most influential feature, accentuates the community's acknowledgment of a developer's input. A high 'tag_score' signifies domain-specific expertise. Metrics like 'average_score_per_answer' and 'first_answers' encapsulate both the caliber and the regularity of a user's contributions—the former indicating consistent answer quality and the latter reflecting active community participation. 'Accepted_answers' further vouch for the quality and pertinence of a developer's knowledge dissemination. While GitHub metrics provide a window into a developer's coding habits, Stack Overflow metrics delve into their community engagement and problem-solving acumen. Collectively, insights from both platforms paint a holistic picture of a developer's expertise in a specific technology.

4.1 DISCUSSION

Our results underscore the potential of integrating data from multiple platforms to enhance the precision of evaluating developers' expertise, especially those involved in serverless functions. The incorporation of Stack Overflow features alongside GitHub data, particularly in the context of serverless function development, has shown promising improvements in our model's performance.

In the industrial landscape, our model suggests a nuanced approach to recruitment, talent acquisition, and team optimization, especially for roles centered around serverless functions. By amalgamating coding practices and community engagement metrics from platforms like GitHub and Stack Overflow, there's potential for a more in-depth insight into a developer's skills and contributions in the serverless domain. Such insights could refine the hiring process, potentially leading to more targeted training and role assignments. Although our study's focal point is the serverless function domain, the method might be adaptable to other technological areas. For new instances, our model provides a framework for assessing individual developer profiles, potentially offering predictions on their expertise levels based on the features we've identified.

In the academic realm, our model could serve as a potential reference for research, curriculum development, and student assessment in serverless function development and beyond. It might offer insights for studies exploring developer behavior and proficiency, especially when integrating data from multiple sources. By evaluating students' real-world coding activities, there's an opportunity for educators to offer feedback that encompasses both theoretical knowledge and practical application, promoting a balanced learning experience.

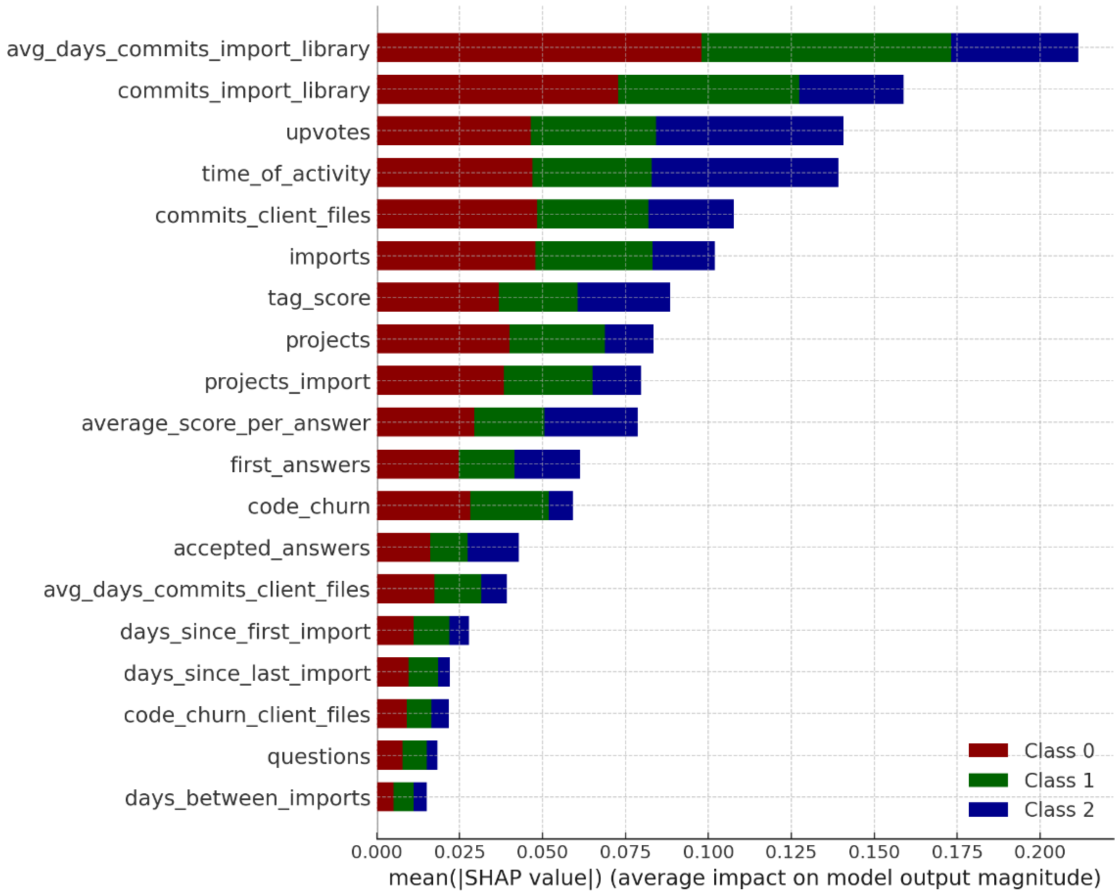


Figure 7: SHAP Summary Plot

5. Threats to Validity

Research, especially in the domain of empirical studies, is often subject to various threats that might affect the generalizability and validity of the results. In this section, we discuss potential threats to the validity of our study and the measures we've taken to mitigate them.

Model Bias: While we experimented with multiple machine learning algorithms, each model comes with its inherent biases. For instance, tree-based models like Random Forest might overfit on certain datasets, while linear models like Logistic Regression might not capture non-linear relationships effectively.

Hyperparameter Tuning: Although we employed GridSearchCV for exhaustive hyperparameter tuning, there's always a possibility that a different combination might yield slightly better results.

Feature Selection: The inclusion or exclusion of features can impact model performance. Our hybrid feature selection method combined heuristic labeling with kNN label imputation. However, heuristic labeling, despite aiding dataset balance, may not always reflect the intricacies of developer expertise. Likewise, while kNN is widely used, it can be susceptible to mislabeling due to noisy neighbors.

Dataset Specificity: Our study is based on specific datasets tailored to certain programming languages. The findings might not be directly generalizable to other languages or platforms.

Data Absence from Other Platforms: The lack of data from platforms like LinkedIn and TopCoder may limit our model's comprehensiveness. Missing insights from these platforms could challenge the external validity of our findings, potentially overlooking essential indicators of developer expertise.

Labeling and Classification: The classification of developers into categories like Novice, Intermediate, and Expert is based on certain metrics and might not capture the complete essence of a developer's expertise.

Feature Interpretation: While we employed the SHAP method for feature importance analysis, the interpretation of the importance of certain features might vary among experts.

Self-Assessment Accuracy: While we operated under the assumption that the self-assessments provided by contributors were accurate reflections of their expertise levels, there's an inherent risk associated with relying on subjective evaluations. Contributors might have overestimated or underestimated their skills due to factors like overconfidence, modesty, or a lack of clear understanding of the assessment criteria. This potential discrepancy between perceived and actual expertise could influence the validity of our findings, especially if these self-assessments were used as ground truth or reference points in our analysis.

6. Conclusion and Future Directions

In this research, we ventured into the domain of predicting developer expertise specifically within the realm of serverless functions, using features extracted from GitHub and Stack Overflow. Our findings underscored the value of multi-platform data integration in providing an in-depth understanding of developer expertise.

In our research, we explored two key areas. The first area of exploration (RQ1) revolved around identifying the most effective machine learning algorithms for evaluating developer expertise. Our journey led us to the Random Forest (RF) model, which consistently demonstrated robust performance across multiple datasets. We also observed commendable performance from the Support Vector Machine (SVM) model, particularly in terms of precision.

The second area of exploration (RQ2) focused on uncovering the features or metrics that best indicate a developer's expertise in serverless functions. Our exploration revealed that the top 5 indicators were 'avg_days_commits_import_library', commits_import_library', 'upvotes', 'time_of_activity', and 'commits_client_files'. We used the SHAP method for feature importance analysis to arrive at these insights.

Our research adds to the field by predicting developer expertise in serverless functions, an area not widely studied before. We used 22 features from GitHub and Stack Overflow, which is more than what's typically used in this domain. This large set of features gives us a detailed look at developer activities and expertise. While there are other studies [4], [5] that also use data from multiple platforms, our study stands out because we use both GitHub and Stack Overflow data and a larger set of features. Our results agree with other studies that find it useful to combine insights from multiple platforms. But our research goes one step further by showing how this approach works well for serverless functions.

In addition, one of the tangible outputs of our research is the creation and public release of six language-specific datasets, representing our target languages. By making these datasets publicly available[8], we not only aim to contribute to the academic community but also hope to foster further research in this area.

Our current investigation has also highlighted the potential benefits of integrating insights from platforms such as Stack Overflow. Such an integrative approach, which merges data from varied sources, can offer richer insights into the multifaceted nature of developer expertise, especially in the context of serverless functions.

[8] https://github.com/aref98/Evaluating-Developer-Expertise-in-Serverless- Functions-by-Mining-Activities-from-Multiple-Platforms

As we move forward, there are multiple avenues we can explore to build upon our current findings. One potential direction is to widen our data collection to encompass a more extensive range of repositories, offering a deeper dive into developer activities. Another promising avenue is to investigate other metrics of developer expertise, such as peer reviews or code quality assessments, which might yield a more nuanced understanding. Finally, considering the vast ecosystem of developer platforms, integrating data from platforms like GitLab, Bitbucket, and LinkedIn, as well as competitive coding platforms like TopCoder, can provide a more rounded view of developer behavior and skills. This could be the next step in further refining and expanding our understanding of expertise in serverless function development.

7. References

[1] S. Kourtzanidis, A. Chatzigeorgiou, and A. Ampatzoglou, "RepoSkillMiner: Identifying software expertise from GitHub repositories using Natural Language Processing," *Proc. - 35th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE, 2020*, pp. 1353–1357.

[2] X. Song, J. Yan, Y. Huang, H. Sun, and H. Zhang, "A Collaboration-Aware Approach to Profiling Developer Expertise with Cross-Community Data," *IEEE Int. Conf. Softw. Qual. Reliab. Secur. QRS*, Guangzhou, China, 2022, pp. 344–355.

[3] E. Constantinou and G. M. Kapitsaki, "Developers expertise and roles on software technologies," *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, Hamilton, New Zealand, 2016, pp. 365–368.

[4] Y. Tian, W. Ng, J. Cao, and S. McIntosh. (2019, Nov.). Geek talents: Who are the top experts on GitHub and stack overflow?. *Comput, Mater & Contin.* [Online]. 61(2), pp. 465–479. Available: 10.32604/cmc.2019.07818

[5] S. L. Vadlamani and O. Baysal, "Studying Software Developer Expertise and Contributions in Stack Overflow and GitHub," IEEE International Conference on Software Maintenance and Evolution (ICSME), Adelaide, SA, Australia, 2020, pp. 312–323.

[6] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski. (2019, Nov.). The rise of serverless computing. Communications of the ACM. [Online]. 62(12), pp. 44–54. Available: 10.1145/3368454

[7] M. Shahrad et al., "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," Proc. 2020 USENIX Annu. Tech. Conf. ATC, 2020, pp. 205–218.

[8] A. Mujezinovic and V. Ljubovic, "Serverless architecture for workflow scheduling with unconstrained execution environment," 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatiapp, 2019, pp. 242–246.

[9] R. Cordingly et al., "Implications of Programming Language Selection for Serverless Data Processing Pipelines," IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Calgary, AB, Canada, 2020, pp. 704–711.

[10] J. Oliveira, M. Viggiato, and E. Figueiredo, "How well do you know this library? Mining experts from source code analysis," *ACM Int. Conf. Proceeding Ser.*, 2019, pp. 49-58.

[11] J. E. Montandon, L. Lourdes Silva, and M. T. Valente, "Identifying experts in software libraries and frameworks among GitHub Users," *IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, Montreal, QC, Canada, 2019, pp. 276–287.

[12] B. Vasilescu, V. Filkov, and A. Serebrenik, "StackOverflow and GitHub: Associations between software development and crowdsourced knowledge," *International Conference on Social Computing, Alexandria, VA, USA*, 2013, pp. 188–195.

[13] A. Santos, M. Souza, J. Oliveira, and E. Figueiredo, "Mining software repositories to identify library experts," ACM Int. Conf. Proceeding Ser., 2018, pp. 83–91.

[14] G. J. Greene and B. Fischer, "CVExplorer: Identifying candidate developers by mining and exploring their open source contributions," Proceedings of the 31st IEEE/ACM international conference on automated software engineering, 2016, pp. 804–809.

[15] X. T. Trinh, "Online learning of multi-class Support Vector Machines," Master dissertation, no. 12 061, 2012.

[16] Christopher J.C. Burges. (1998, Jun.). A Tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery. [Online]. 2(2), pp. 121–167. Available: 10.1023/A:1009715923555

[17] G. Biau and E. Scornet. (2016, Apr.). A random forest guided tour. Test. [Online]. 25(2), pp. 197–227. Available: 10.1007/s11749-016-0481-7

[18] A. Natekin and A. Knoll. (2013, Dec.). Gradient boosting machines, a tutorial. Front. Neurorobot. [Online]. 7, p. 21. Available: 10.3389/fnbot.2013.00021

[19] A. Schneider, G. Hommel, and M. Blettner. (2010, Nov.). Lineare regressionsanalyse - Teil 14 der serie zur bewertung wissenschaftlicher publikationen. Deutsches Ärzteblatt International. [Online]. 107(44), pp. 776–782. Available: 10.3238/arztebl.2010.0776

[20] Ö. Senger. (2013, Aug.). Impact of skewness on statistical power. Modern Applied Science. [Online]. 7(8), pp. 49–56. Available: 10.5539/mas.v7n8p49

[21] N. J. Gogtay and U. M. Thatte. (2017, Mar.). Principles of correlation analysis. Journal of the Association of Physicians of India. [Online]. 65(3), pp. 78–81.

[22] S. Chulani, B. Boehm, and B. Steece. (1999, Jul.). Bayesian Analysis of Empirical Software Engineering Cost Models. IEEE Transactions on Software Engineering. [Online]. 25(4), pp. 41–51. Available: 10.1109/32.799958

[23] J. Li et al. and H. Liu. (2017, Dec.). Feature selection: A data perspective. ACM computing surveys (CSUR). [Online]. 50(6), pp. 1-45. Available: 10.1145/3136625

[24] B. Boecking, W. Neiswanger, E. P. Xing, and A. Dubrawski, "Interactive Weak Supervision: Learning Useful Heuristics for Data Labeling," ICLR 2021 - 9th Int. Conf. Learn. Represent., 2021, pp. 1–27.

[25] S. Zhang, (2012, Nov.). Nearest neighbor selection for iteratively kNN imputation. Journal of Systems and Software. [Online]. 85(11), pp. 2541–2552. Available: 10.1016/j.jss.2012.05.073

[26] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online Defect Prediction for Imbalanced Data," IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italyvol, 2015, pp. 99–108.

[27] L. Li, T. F. Bissyandé, D. Octeau, and J. Klein, "Reflection-aware static analysis of android apps," Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, 2016, pp. 756–761.

[28] A. S. Singh, M. B. Masuku, and Department. (2014, Nov.). Sampling techniques & determination of sample size in applied statistics research. International Journal of economics, commerce and management. [Online]. 2(11),

pp. 32–33. Available: ijecm.co.uk

[29] L. Buitinck et al. and R. Layton, "API design for machine learning software: experiences from the scikit-learn project," European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases, 2013, pp. 1–15.

[30] R. M. Dawes, "The robust beauty of improper linear models in decision making.," Rationality and Social Responsibility, 2008, pp. 321-344.

[31] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," Adv. Neural Inf. Process. Syst., 2017, pp. 4766–4775.

[32] A. Sayers, Y. Ben-Shlomo, A. W. Blom, and F. Steele. (2016, Jun.). Probabilistic record linkage. International Journal of Epidemiology. [Online]. 45(3), pp. 954-964. Available: 10.1093/ije/dyv322