Empirical Analysis and Enhancement of Machine Learning Software Documentation

SHARUKA PROMODYA THIRIMANNE

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

YORK UNIVERSITY
TORONTO, ONTARIO

October 2024

Abstract

As machine learning gains popularity, individuals from diverse fields and skill levels integrate it into their workflows. However, many lack software engineering experience, impacting the usability of documentation. Additionally, the current machine learning documentation and its issues are insufficiently addressed in the literature. This thesis comprises two papers. In the first paper, we compared the content and design differences of TensorFlow tutorials and analyzed the profiles of users who asked questions about them. We also developed a comprehensive taxonomy of TensorFlow documentation issues. In the second paper, we examined the potential of leveraging generative AI to augment machine learning documentation. We proposed a method to augment TensorFlow API documentation by addressing documentation-related questions using large language models. This thesis highlights the need for machine learning documentation to accommodate diverse skill levels as its use expands across domains and showcases the potential of generative AI to automate documentation augmentation.

Acknowledgements

The completion of this thesis would not have been possible without the support, guidance, and encouragement of many individuals, to whom I am deeply grateful. First, I sincerely thank my supervisors, Prof. Maleknaz Nayebi and Prof. Suprakash Datta, for their unwavering support and guidance throughout my graduate studies. Their expertise, patience, and encouragement have been invaluable, and I am grateful for the mentoring that has significantly contributed to my academic and personal growth.

I express my deepest gratitude to my beloved family and friends for their love, encouragement, and support throughout this journey. Their unwavering backing, understanding, and patience have been a constant source of motivation.

I am thankful to the department for providing the resources and facilities necessary for my research. Their support has been crucial in helping me navigate the challenges of graduate studies. Finally, I extend my deepest gratitude to my lab mates for their unwavering support, collaboration, and friendship. Their passion and commitment to research have been a source of inspiration, and I have gained invaluable knowledge from them.

Table of Contents

List of Figures

List of Tables

Abbreviations & Acronyms

OSS	**O**pen **S**ource **S**oftware
SE	**S**oftware **E**ngineering
ML	**M**achine **L**earning
DL	**D**eep **L**earning
LLM	**L**arge **L**anguage **M**odel
RAG	**R**etreival **A**ugmented **G**eneration
API	**A**pplication **P**rogramming **I**nterface
SO	**S**tack **O**verflow

Chapter 1

Introduction

Software documentation is a formal writing that supports the efficient and effective use of software in its intended environment [21] and it is considered an integral part of the software development life cycle [126]. It offers essential guidance to developers on the proper utilization of a tool or service, providing information about the design, code, interfaces, and functionality of software [4]. Various artifacts, such as comments, reviews, pull requests, release notes, commit messages, and API documents, clarify code functionality and reflect changes over time in natural language [152, 161] to support developers. Traditionally, software developers create documentation for their code primarily for other developers working in the same field.

Software documentation has been extensively analyzed in the literature, detailing the problems developers encounter when working with it [37, 138, 170, 77]. Moreover, studies identified common issues in software documentation, including insufficient or outdated content, unclear information, and lack of examples [6]. Documentation of machine learning software is not an exception. Poor documentation of machine learning components can hinder usefulness, slow down time to market, and impede adoption [61]. As machine learning continues to gain popularity, individuals from diverse fields are integrating it into their workflows through programming. However, many of these users lack software engineering experience, which impacts the usability of the documentation. The growing application of software code in machine learning means that

many users who depend on this documentation are not professional software developers, making it challenging for them to interpret and effectively utilize these resources.

1.1 Stack Overflow for Documentation-Related Concerns

Developers frequently seek assistance and advice for technical challenges by posting questions on Stack Overflow. The platform moderates hundreds of thousands of posts each month from developers with diverse backgrounds and questions on a wide range of topics. Consequently, Stack Overflow serves as a comprehensive knowledge repository of developers' needs [22]. Utilizing Stack Overflow to identify developers' questions and pain points is a well-established study design in software engineering [7, 109]. Specifically, this approach has been employed to identify issues in software documentation [5, 165, 19, 128]. Aghajani et al. [5] conducted a quantitative study using mailing lists, Stack Overflow, and project repositories to identify issues developers encountered with software documentation. They developed a comprehensive taxonomy comprising 162 types of documentation issues related to the information contained within, its presentation, the documentation process, and tool support. They further conducted a survey to compare the mined repository results with developers' perspectives [6].

The majority of these studies were conducted in the domain of software engineering, leaving a gap in similar research within the domain of machine learning, despite its increasing usage. Moreover, various studies highlight the increasing trend of machine learning and Python-related questions on Stack Overflow [184, 10]. In Chapter 2, we addressed this research gap by conducting a comprehensive analysis of Stack Overflow questions at the intersection of machine learning and software engineering, specifically focusing on TensorFlow and Python. We constructed a taxonomy of issues that developers encounter, identifying the nature and causes of documentation-related questions. We selected TensorFlow documentation for two main reasons. Firstly, TensorFlow is recognized as the most utilized machine learning library [162]. Secondly, among the machine learning libraries *Caffe, Keras, Mahout, MLlib, Scikit-learn, Theano*, and *Torch*, TensorFlow has the highest number of highly-rated posts on Stack Overflow

[70]. Moreover, an analysis of Stack Overflow tags associated with TensorFlow reveals that Python is the most frequently used tag, appearing 1.9 times more often than the second most popular tag, "Keras", and 48.7% more frequently than Java.

1.2 Current Documentation and Developer Skill Levels

Software applications are continuously expanding their scope and are used across diverse domains, from life sciences and economics to the fashion industry and manufacturing. Therefore, many developers are familiar with the application domains that are important for program comprehension [143], but their education and experience in programming and software engineering are quite diverse. Developers rely on their competence, profile, and skill for various software engineering tasks that are crucial for the successful and timely completion of development projects [13, 132]. Thus, for documentation to be effective, it should accommodate users with a wide range of skills, from those with limited knowledge to highly proficient users [58].

User skill levels have been extensively studied within both the software engineering domain, and broader Web and Q/A platforms [3, 41, 111, 28]. For this, most research in software engineering have focused on Stack Overflow [27, 43, 12, 188, 133, 39, 101]. By systematically understanding and differentiating user skill sets, we can design documentation that better matches their competence. Fundamentally, the competence of a user in a specific domain is determined by their experience, knowledge, achievements, and contributions [172].

We analyzed the documentation of popular ML libraries: `Pytorch`, `Scikit-learn`, `TensorFlow`, `Keras`, `Apache MXNet`, `Caffe`, `Theano`, and `CNTK` (Microsoft Cognitive Toolkit), to determine the consideration of user skill levels in their documentation. Among them, TensorFlow is presently the sole machine learning library that has recognized and tailored its documentation to accommodate users of varying skill levels, providing tutorials at two levels: beginner and advanced. Therefore, understanding the relationship between the skill level of developers and the level of documentation they reference enables the creation of effective documentation tailored to users of varying skill levels.

In Chapter 2, we conducted a comparative analysis to distinguish the content and design differences between the two tutorial levels and to examine the backgrounds of the developers who referenced these documents.

1.3 Generative AI for Augment Documentation

Software documentation, which passively meets developers' information needs, has led to the development of automated tools and techniques for generating and augmenting documentation [44, 107, 140]. Automated documentation generation and augmentation have been extensively studied in the literature, primarily focusing on template-based strategies [2, 107, 116], which are constrained by their underlying templates, and information retrieval-based strategies [46, 182, 183] that rely on similarity measures. These approaches include leveraging code examples and reviews from technical Q&A sites [168], using "insight sentences" from Stack Overflow [166], and enriching API documentation by analyzing software change history [15]. However, existing methods often augment documentation without addressing the underlying causes of documentation-related questions.

Large Language Models (LLMs) have shown remarkable performance in a variety of natural language processing (NLP) tasks [192]. Recent advancements, including increased model sizes and larger training datasets, have significantly enhanced their performance in many NLP applications [82]. However, LLMs generate responses that contain factual errors, known as "hallucinations" [89, 33, 135, 71]. These factual inaccuracies arise due to biases in their training data and their limited access to up-to-date, proprietary, or domain-specific information [62, 150]. The introduction of the retrieval-augmented generation (RAG) technique has helped address this issue by improving the model's ability to generate responses with high factual accuracy and relevant contextual information [88, 87]. Moreover, it offers significant advantages by not requiring knowledge to be stored in model parameters. Instead, knowledge can be provided to the LLM

in a plug-and-play manner, ensuring remarkable scalability. Furthermore, this knowledge can be sourced from both supervised datasets with input-output pairs and unstructured data from unsupervised corpora [87]. This enables the augmentation of API documentation with factually accurate information by incorporating domain-specific contextual information relevant to addressing documentation-related questions. The question-and-answer forum Stack Overflow serves as a repository of API knowledge , offering "how-to" documentation in response to specific needs [93] and addressing undocumented issues along with their solutions [58]. This makes it an ideal source for contextual information retrieval for RAG.

Furthermore, studies on code generation using LLMs have demonstrated that employing an iterative approach that repeatedly executes and refines generated code significantly improves code accuracy [137]. Recent research has also introduced zero-resource hallucination detection techniques to further mitigate hallucinations by prompting the LLM multiple times [104]. Additionally, advancements in prompt engineering techniques have shown that zero-shot chain-of-thought prompting improves the reasoning capabilities of LLMs [82].

Leveraging the capabilities of generative AI in reasoning, generating factually accurate responses based on contextual information, and iterative code generation, we propose in Chapter 3 an automated machine learning API documentation augmentation tool called DocChameleon. This tool augments TensorFlow API documentation by addressing documentation-related questions on Stack Overflow.

1.4 Thesis Overview

This thesis consists of three chapters following the introduction.

In Chapter 2, we present our paper titled **"Current Trends in TensorFlow and the Future of ML Software Documentation: One Documentation Does Not Fit All"**. This paper is submitted to the Journal of Systems and Software (JSS). We conducted an exploratory empirical study situated at the intersection of software engineering and machine learning, with a focus on Python and TensorFlow to evaluate current machine

learning documentation [181]. Initially, we compared the content of TensorFlow beginner and advanced tutorials, examining their topics, readability, word and sentence count, and technicality. We then investigated the nature and causes of issues related to TensorFlow documentation on Stack Overflow. Finally, we assessed whether the levels of TensorFlow tutorials align with the skill levels of developers by comparing Stack Overflow questions on beginner and advanced level tutorials using several state-of-the-art metrics.

In Chapter 3, we present our second paper titled **Retrieval-Augmented Editing of TensorFlow API Documentations based on Stack Overflow Questions**. In this paper, we introduce an automated machine learning API documentation augmentation tool called DOCCHAMELEON, which augments TensorFlow API documentation by addressing documentation-related questions on Stack Overflow using LLMs and the RAG technique. DOCCHAMELEON augments API documentation with (1) generated explanations, addressing documentation ambiguities as users often request explanations to resolve those ambiguities, (2) generated code examples, since a majority of documentation-related issues arise due to the lack of code examples, (3) links to relevant YouTube tutorials, as developers frequently turn to YouTube for gaining API knowledge [47], and (4) links to related Stack Overflow Q&A posts, as Stack Overflow is widely recognized as a valuable resource for learning APIs by offering API knowledge through a combination of textual descriptions and code examples. Additionally, these resources address various undocumented issues that developers encounter in practice, along with their solutions [186].

In Chapter 4, we conclude our study by summarizing our key findings and contributions, and discussing the potential for future research.

Chapter 2

Current Trends in TensorFlow and the Future of ML Software Documentation: One Documentation Does Not Fit All

Abstract

Software documentation provides guidance on the proper use of tools or services. With the rapid growth of machine learning libraries, individuals from various fields are incorporating machine learning into their workflows through programming. However, many of these users lack software engineering experience, affecting the usability of the documentation. Traditionally, software developers have created documentation primarily for their peers, making it challenging for others to interpret and effectively use these resources. In this study, we examined customization trends in TensorFlow tutorials and compared these artifacts to analyze content and design differences. We also analyzed Stack Overflow questions related to TensorFlow documentation to understand the types of questions and the backgrounds of the developers asking them. Further, we developed

two taxonomies based on the nature and triggers of the questions for machine learning software.

Our findings showed no significant differences in the content or the nature of the questions across different tutorials. Our results show that 24.9% of the questions related to TensorFlow documentation concern errors and exceptions, while 64.3% relates to inadequate and non-generalizable examples in the documentation. Despite efforts to create customized documentation, our analysis indicates that current TensorFlow documentation does not effectively support its target users. We discuss the future of machine learning documentation and the potential of generative AI to improve software artifacts.

2.1 Introduction

Software code is commonly documented by software developers for other software developers in the same domain. However, with the growing use of software code in various application fields such as Machine Learning (ML), many who use the code, hence needing the code documentation, may not necessarily be software developers by profession. Code documentation, spanning various artifacts such as comments, reviews, pull requests, release notes, commit messages, and API documents, clarifies code functionality and reflects changes over time in natural language [152, 161]. Studies identified common issues in software documentation, including insufficient or outdated content, unclear information, and lack of examples [6].

As software progressively dominates the world, its application is getting broader in various domains, ranging from life sciences to economics to the fashion industry to manufacturing. Such developers are rather familiar with the application domain, while their educational and experience background is quite diverse when it comes to programming (i.e., they may know numerous programming languages and have experience in various domains in which programming is applied) and, more broadly, software engineering. This diversity necessitates accessible and understandable documentation tailored to different user needs. Skilled professionals have been relied upon for a variety of software engineering tasks [14, 146, 132], and their competence, profile, and

skill are crucial in successfully completing development tasks promptly [13, 132, 14]. We argue for documentation to be effective, it should faithfully represent the software while matching users' skills and proficiency [58]. Documentation of machine learning software is not an exception.

2.1.1 Motivation

Machine learning software developers, system integrators, and end users come from diverse backgrounds, creating a varied user profile. Poor documentation of machine learning components can hinder usefulness, slow down time to market, and impede adoption [61]. Ambiguous, outdated, and misleading documentation forces users to seek other resources, such as Stack Overflow, a question-and-answer (Q&A) community. Several studies investigated Stack Overflow's role and usefulness (see, for example, [184, 83]) and the increasing trend in question concerning machine learning and, generally, Python [10, 59].

Motivated by these challenges and aligned with the efforts on Software Engineering for Machine Learning (SE4ML) [79, 105], we argue that the future of software documentation is to further align with the cross-disciplinary skills of machine learning users and even further to adjust and personalize this documentation to the use of individual developers. The potential of LLMs and generative AI to revolutionize software documentation and development practices is immense. Embracing these technologies means transitioning from viewing development tools as mere facilitators to treating them as intelligent partners in the software creation process. This shift promises not only to enhance productivity and innovation but also to redefine the boundaries of what is possible in software development [125, 49].

Hence, we conducted an exploratory empirical study to assess current machine learning documentation [181]. Our paper aims to first examine the challenges developers face using machine learning documentation and further evaluate the extent to which the current documentation matches users' skill sets and experience. This research benefits developers, software engineers, and researchers developing automated software documentation models (e.g., [116, 66]) by providing insights into the challenges faced by

machine learning users and providing insights into effective documentation that caters to the diverse needs of users.

Based on our observations, we argue that the substantial discrepancy between user proficiency and documentation complexity needs special attention. This gap can lead to frustration, which is inefficient and far from an optimal use of time. Based on our findings, we explore the potential of generative AI for designing and implementing software documentation and envision the future of software documentation to be more personalized for developers.

In Section 2.2, we outline our case study design for TensorFlow documentation, exploring the problem space [181]. Section 3.4 details the empirical data used. In Section 2.4, we synthesized a taxonomy of identified problems and causes and extended our analysis to explore disparities between TensorFlow beginner and advanced documentation. Furthermore, we investigated how the current state of practice caters to user questions among individuals with varying experiences and reputations on Stack Overflow. Section 3.7 presents the study findings. Potential threats to the validity of our study are addressed in Section 2.6. Subsequently, the implications of our findings and the future works are explored in Section 3.8. Finally, related work is discussed in Section 2.8 followed by the conclusion in 3.10.

2.2 Study Design

Our study is situated at the intersection of software engineering and machine learning with a focus on Python and TensorFlow. We aimed to understand the challenges developers face when using such documentation. Here, we discuss the context and protocols for our case study.

2.2.1 Case study Context Selection

We conducted a case study at the intersection of machine learning (TensorFlow) and software engineering (Python). Our choice was motivated by two main reasons:

First, We manually and thoroughly looked into the documentation of popular ML libraries `PytorchTorch`, `Scikit-learn`, `TensorFlow`, `Keras`, `Apache MXNet`, `Caffe`, `Theano`, and `CNTK` (Microsoft Cognitive Toolkit). TensorFlow is the only machine learning library that considers its users' skill level in its tutorials. TensorFlow provides three documentation types, including Tutorials, API Documentation, and Community Translations. The tutorials section has two levels: beginner and advanced. There is no detailed explanation of the difference between these two, while it considers the beginner tutorials the "best place to start with user-friendly sequential API" and the advanced tutorials "provides a defined by-run interface for advanced research"[158]. Analysis of Stack Overflow tags associated with TensorFlow shows that Python is the most frequently used tag, with a frequency 1.9 times higher than the second most popular tag "Keras", and 48.7% higher than Java. (See Figure 2.1 for a word cloud of the tags.)

Second, TensorFlow is a widely used machine learning library that was originally developed by `Google` and is now available as an open-source project. It provides wrappers in both Java and Python. Python is the preferred language for data science and machine learning [134], and it is one of the supported programming languages for TensorFlow.

Notably, TensorFlow is currently the only machine learning library adopting its documentation (even though partially and only for its tutorials) to the skill level of its users (beginner and advanced).

Hence, we instantiate our case study by analyzing the nature of TensorFlow tutorials which are part of the TensorFlow documentation echo system.

2.2.2 Research Questions

Of all the available machine learning tools and libraries, only TensorFlow has considered different types of documentation for beginner and advanced developers, *implicitly* recognizing the need for different documentation. Hence, using this context, we answer three research questions. Through this case study, we investigate the below RQs;

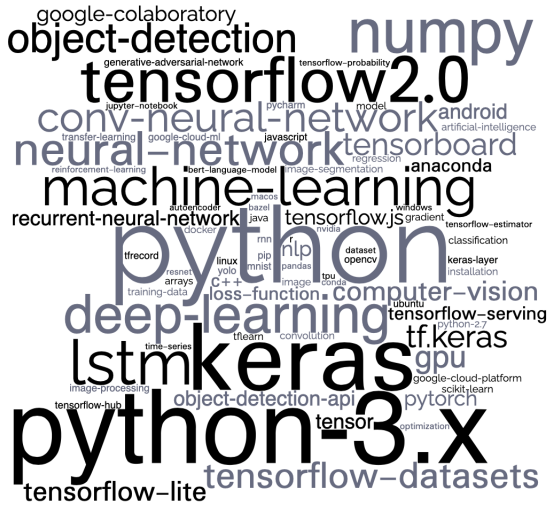


FIGURE 2.1: Overview of the hashtags around TensorFlow

RQ1: How do the content of TensorFlow beginner and advanced tutorials compare? We focus on the TensorFlow tutorials that explicitly differentiate between beginner and advanced developers. We compare these in terms of the topics and subjects covered, the readability, word and sentence count of their content, and technicality (examples count and code snippets' size).

RQ2: How has the documentation for TensorFlow been discussed on Stack Overflow? We investigate whether users of TensorFlow face difficulties with the library's documentation and, if so, identify the nature and causes of these issues on the popular Q&A platform, Stack Overflow. We performed a systematic manual analysis to create taxonomies of the types of problems encountered and the root causes of these issues.

RQ3: How do the years of experience and reputation of developers on Stack Overflow compare when asking questions about beginner versus advanced tutorials? To assess if TensorFlow-defined tutorial levels align with developers' skills, we compared those who posted questions on Stack Overflow about either beginner or advanced tutorials using several state-of-the-art metrics.

2.3 Empirical Data

We gathered and used the TensorFlow tutorials, designed for beginner and advanced users by the provider, Google, to answer **RQ1**. Then we gathered questions about TensorFlow documentation from Stack Overflow to answer **RQ2**, and then retrieved the user profiles of those who asked questions about TensorFlow tutorials to answer **RQ3**. The process steps are shown in Figure 3.3, and we refer to those steps in what follows. The data gathering steps were annotated with yellow circles (x).

2.3.1 TensorFlow Tutorials (RQ1)

TensorFlow provides tutorials for three types of artifacts: CORE, HUB, and ADD-ONS. Only the CORE product has tutorials for two competence levels: "beginner" and "advanced". Although TensorFlow officially maintains tutorials for HUB and ADD-ONS, they did not distinguish between different levels of competence. Therefore, for our analysis of **RQ1**, we focus specifically on the TensorFlow CORE product tutorials. To gather these tutorials in Step (1), we developed a Python scraper using the `Beautifulsoup` library to extract data from the TensorFlow website [159]. We then parsed the data to exclude the special characters (Step (2)), page footers and headers, and ads and gathered the HTML tags for code snippets. We also extracted the Table of Contents for each tutorial, preserving the hierarchy of topics. For each tutorial, we recorded the headers, sub-headers, content, and any hyperlinks, as well as the associated expertise level (beginner or advanced) provided by TensorFlow.

2.3.2 Documentation Related Questions (RQ2)

Using the StackExchange Data Explorer, we gathered questions and user data from Stack Overflow. We did this in two main stages, (i) identifying the keywords that are highly associated with software documentation in Stack Overflow questions, and (ii) using keywords to retrieve questions and manually identifying documentation related questions to form taxonomies.

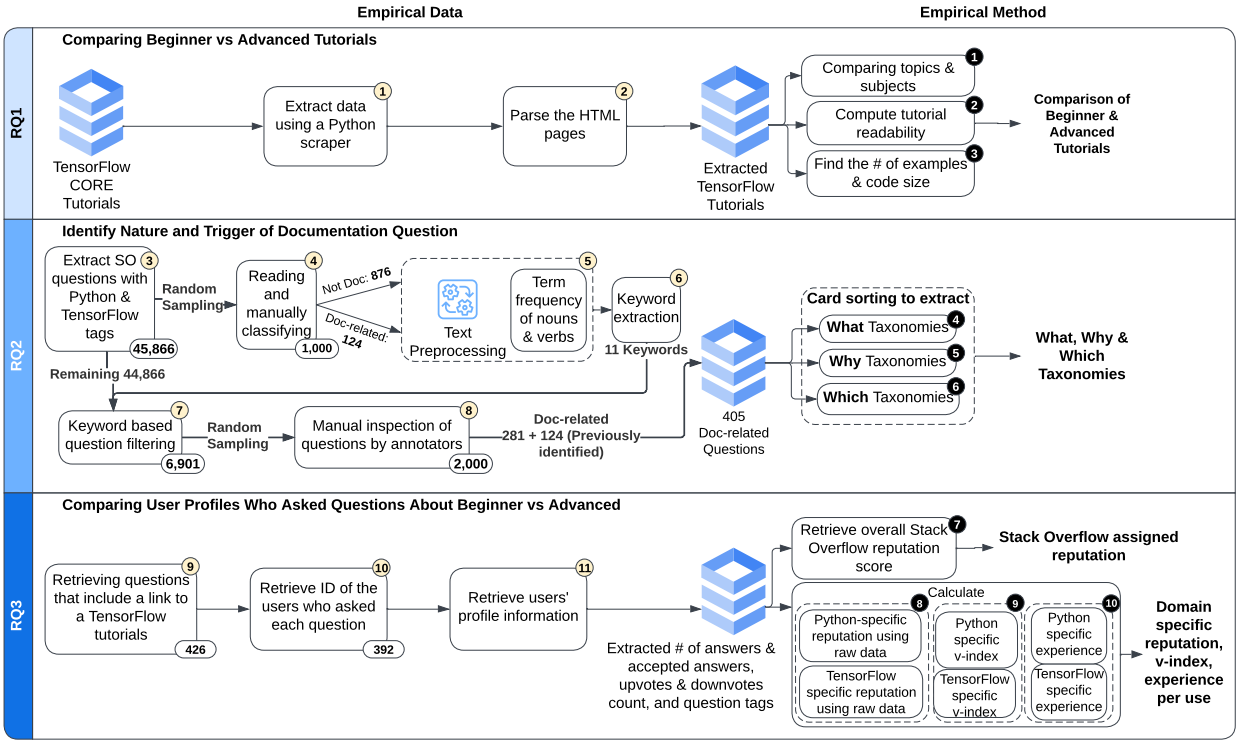


FIGURE 2.2: Empirical Data collection (numbered in yellow) and research methodology (in black) for all the RQs

In Step ③, we queried questions with both the TensorFlow and Python tags and retrieved 45,866 questions. To focus on the documentation-related questions, we randomly sampled 1,000 questions among these. Then (Step ④), two researchers manually inspected the questions by fully reading the question posted on Stack Overflow and separated documentation-related questions from non-documentation ones. Among these 1,000 questions, we identified 124 questions as documentation-related. To increase the documentation-related questions count for our analysis, we extracted keywords frequently occurring in the questions about documentation. To do so, in Step ⑤ we preprocessed data by removing code blocks from the corpus using HTML tags. We then removed standard `NLTK` stop words and numbers from the text. Then, we identified the Part-Of-Speech (POS) tags using `NLTK` and kept only nouns, adjectives, and verbs. The remaining words were then lemmatized, and calculated their TF-IDF values. We repeated these steps for the 876 non-documentation-related questions and ranked the words using their term frequency. We then excluded the frequently occurring words

common between the two lists. To reverse the lemmatization, we used text surface realization with `pynlg`. As a result, in Step (6), we identified frequently used words in documentation-related questions: *documentation, document, doc, guide, tutorial, paper, instruction, official, lecture, book, website.*

Then, we used these keywords to identify questions about documentation among the 44,866 Stack Overflow questions with Python and Tensorflow tags. Hence, in Step (7), we searched for questions with at least one of the documentation-related keywords above. We retrieved 6,901 questions that included at least one of these keywords in their text. We then randomly sampled 2,000 of these questions for manual analysis. In Step (8), two annotators independently read and evaluated whether these questions are relevant to documentation. They identified 281 questions relevant to documentation. Having 124 documentation-related questions from our initial sampling in Step (4), we carried out our **RQ2** with an overall 405 documentation-related questions tagged with both TensorFlow and Python.

2.3.3 Stack Overflow User Profiles (RQ3)

The Stack Exchange API provides endpoints to extract Stack Overflow user information. In Step (9), among those questions, we identified 426 questions that explicitly included a link (URL) to TensorFlow tutorial documentation. In Step (10), we retrieved the user profile of developers asking these questions. Out of 426 questions, we identified 392 unique user IDs (Only 33 users with more than one question). Finally, in Step (11) for each user we retrieved a range of data, including the post IDs, questions' creation date, link to the posts, number of answers, number of accepted answers, up-votes & down-votes count for the question, answers, and the owner ID.

2.4 Empirical Method

In this section, we explain our methodological steps. These steps are annotated with ⓧ in Figure 3.3.

2.4.1 Content of TensorFlow's Beginners vs. Advanced Documentation (RQ1)

We compared the readability score, total sentence and word count, number of examples, and code size for both beginner and advanced TensorFlow tutorials created by the TensorFlow community.

Topic coverage: In Step ❶, when gathering data and comparing the "Beginner" and "Advanced" TensorFlow tutorials (Section 2.3.1 & Table 2.2), we compared the titles and subtitles. We further manually analyzed the content of the tutorials by two independent authors of the paper and presented any similarities in the content and topics.

Text readability: It is proved that academic and technical writing that targets expert users is inherently more complex in reading compared to the general texts in magazines or Wikipedia targeting non-experts [60]. Hence, to deepen our analysis of TensorFlow tutorials, we evaluated if beginners' tutorials are more readable compared to advanced ones. In Step ❷, we used the Flesch Reading Ease score as a widely used metric for readability [51]. The Flesch score ranges from 0 to 100, where the higher score indicates a text is easier to read. Flesch Reading Ease assumes that short sentences and words with fewer syllables are easier to understand. Here, a [0-30] score indicates a hard-to-comprehend text, while plain English normally scores in [60-70]. We used `Textstat`[1] `Python` package to calculate Flesch Reading Ease for each tutorial.

Number and size of code examples: Different tutorials have a different level of technicality based on the topic they cover [30, 155]. The number of technical examples could indicate the level of technicality in code documentation [52]. In Step ❸, we counted the number of examples in a tutorial based on the descriptions right above each code block using the keyword "example". Then, one of the authors manually checked these identified examples and further scanned the tutorials to verify the identified example counts in TensorFlow tutorials. We further retrieved the size of code snippets in the tutorial by counting the lines of code provided in between the HTML `<code>` tags.

[1] https://pypi.org/project/textstat/

2.4.2 Nature and Trigger of Documentation Questions (RQ2)

We explained in Section 2.3.2 that as a result of multiple manual and automated steps, we identified 405 documentation-related questions. To answer **RQ2**, two annotators independently and manually analyzed the nature of the problems that users faced regarding TensorFlow documentation, as well as the origins of these problems that caused the questions. In each case, we defined what documentation the user is questioning. In our annotations, we answered:

What types of problems do developers encounter when using TensorFlow documentation?

Why did the issues occur when users were working with the TensorFlow library?

Which type of documentation was being discussed or referenced in the question?

Two annotators separately answered each of the above questions. Both annotators have over a year of industry software engineering experience and M.Sc. degrees in Software Engineering. To answer the “What" question and explore the nature of the problem, we used the defined categories by Islam et al. [70]. These categories defined 27 different types of questions commonly asked by Stack Overflow users about machine learning. In Step ❹, each developer conducted a closed card sorting [195] to assign one of these categories to each question to identify the nature of the problem and answer the “What" question. Then, a moderator identified the disagreements and resolved them to finalize issues in each category.

In Step ❺, to determine the cause of the problems and respond to the “why" question, we analyzed the entire question thread, including both the original question and any responses manually. In this case, two annotators independently performed an open card sorting [195] where each independently analyzed the question threads and grouped them based on the similarity in the causes of the problem. Then, they named each group with a title representative of the content. In this process, they could create

new categories as needed. They then discussed and chose category names best representative of the questions and any disagreements in the process. Finally, the moderator identified and resolved the disagreements.

In Step ❻, we also identified the documentation that was subject to question to answer “which" question and determine whether the user referred to official TensorFlow documentation (such as tutorials or API documentation) or third-party documentation. Although we did not have a predefined set of labels for these categories, they were factual decisions not subject to annotator bias. Therefore, the two annotators collaborated to define the “Which" categories without overlapping each other’s work.

2.4.3 Comparing Users Who Pose Questions about Beginner and Advanced Tutorials (RQ3)

We analyzed the profiles of users who asked questions about TensorFlow tutorials to understand the relationship between the competence level defined by TensorFlow and the developers’ experience and background knowledge. This analysis was done separately for developers’ software engineering and machine learning backgrounds.

A proficient programmer may be unfamiliar with the concepts of neural networks or machine learning. Conversely, an individual with solid technical and theoretical knowledge of machine learning and modeling is not necessarily an expert programmer. In this context, the questions they pose about software documentation could differ. We measured and compared developers based on three metrics in each field of Python programming and Machine learning using TensorFlow:

Months of experience in the field: In Step ⑩, to determine the user’s domain-specific experience, we computed the duration in months from their first Python or TensorFlow question posted date to the current date. This time frame represents their experience in the field in months.

Reputation overall and in the field: Stack Overflow assigns each registered user an overall reputation score (Step ❼) based on their activities and the community’s perception of their quality and importance. We compared these scores among the two

TABLE 2.1: The details of how users boost or lose reputation points

Reputation increasing activities	**# reputation points**
Question upvoted	+5
Answer upvoted	+10
Answer is marked as "accepted"	+15
Reputation decreasing activities	**# reputation points**
Question downvoted	-2
Answer downvoted	-2

groups. In Step ❽, we used the formula defined by Stack Overflow with five attributes to compute in the field reputation [178], allowing users either increase or decrease their reputation. Table 2.1 depicts the activities that boost and deduct reputation points.

We then compared the overall reputations of users who asked questions about beginner or advanced tutorials. And, we computed the reputation using the same method, focusing solely on questions tagged as Python or TensorFlow.

V-index in the field: Inspired by the definition of *h-index* [26] Wang et al. [178] introduced a metric called *v-index* to proxy users' skill level on stack Overflow. The *v-index* measures the *v number of answers* a user has posted, each with at least *v number of upvote counts*. In Step ❾, we calculated the v-index for each developer separately based on their questions with Python or TensorFlow tags. Finally, we used the Mann-Whitney test [148] to assess whether a significant difference exists between the distributions of users asking questions about beginner and advanced tutorials in terms of any of these metrics.

2.5 Case Study Results

We manually inspected the tutorials and code documentation for leading machine learning libraries, including `TensorFlow`, `PyTorch`, `Scikit-learn`, `Keras`, `MXNet`, and `Caffe`. Among them, TensorFlow is presently the sole machine learning library that has recognized and tailored its documentation to accommodate users of varying skill levels.

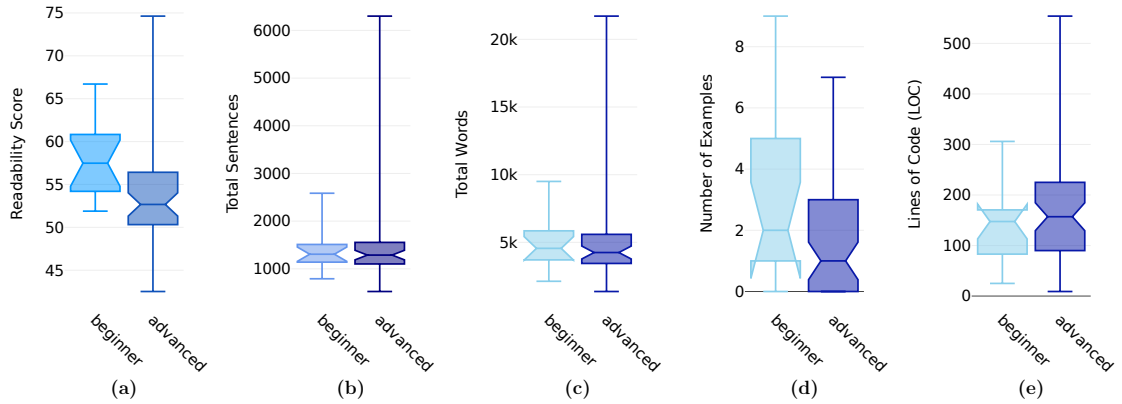


FIGURE 2.3: Distribution of (a) readability score (b) sentence count (c) word count (d) examples count (e) lines of code with respect to the two tutorial levels (beginner and advanced)

Here, we present the findings of **RQ1**, **RQ2**, and **RQ3** to investigate if these skill levels align with the actual TensorFlow's documentation usage.

2.5.1 Tensorflow beginner vs. advanced tutorials (RQ1)

To distinguish between beginner and advanced TensorFlow tutorials, we initially compared their titles and topics. We then investigated the readability of the tutorials as more technical documents are often harder to read [86] and further complemented that with the evaluation of the number of examples and the lines of code in each tutorial. We compared these between beginner and advanced tutorials.

Comparison of titles and topics

TensorFlow offers two beginner and 11 advanced tutorials. We gathered beginner and advanced tutorials of TensorFlow following the process explained in Section 2.3.1. As a result, we identified 70 different topics across 13 tutorial titles. Table 2.2 lists the topics and their competence level as published by TensorFlow. We manually compared the titles, subtitles, and content of these tutorials with two independent researchers.

Table 2.2 has two primary topics within beginner-level tutorials: "ML basics with Keras" and "Load and preprocess data." Additionally, "Distributed training" is one of the 11 topics covered in the advanced-level tutorials. Notably, "ML basics with Keras" (beginner) and "Distributed training" (advanced) share a common sub-topic, under the

TABLE 2.2: TensorFlow tutorials topics, subjects, and the competence level defined by Google.

Title : Beginner	Topic	Count
ML basics with Keras	Basic image classification, Basic text classification, Text classification with TF-Hub, Regression, Overfit & underfit, `Save and load`, Tune hyperparameters with the Keras Tuner	7
Load & preprocess data	Images, Videos, CSV, NumPy, pandas.Dataframe, TFRecord & tf.Example, Added formats with tf.io, Text, Unicode, Subword tokenization	10
Title : Advanced	**Topic**	**Count**
Customization	Tensors and operations, Custom layers, Custom training: walkthrough	3
Distributed training	Distributed training with Keras, Distributed training with DTensors, Using DTensors with Keras, Custom training loops, Multi-worker training with Keras, Multi-worker training with CTL, Parameter server training, `Save and load`, Distributed input	9
Vision	Computer vision, KerasCV, CNN, Image classification, Transfer learning & fine-tuning, Transfer learning with tf-hub, Data augmentation, Image segmentation, Object detection with tf-hub, Video classification, Transfer learning (MoViNet)	11
Text	Text and natural language processing, Get started with Keras NLP, Text and NLP guide	3
Audio	Simple audio recognition, Transfer learning for audio recognition, Generate music with an RNN	3
Structured data	Classify structured data with preprocessing layers, Classification on imbalanced data, Time series forecasting, Decision forest models, Recommenders	5
Generative	Stable diffusion, Neural style transfer, DeepDream, DCGAN, Pix2Pix, CycleGAN, Adversarial FGSM, Intro to autoencoders, Variational autoencoder, Lossy data compression	10
Model optimization	Scalable model compression with EPR, TensorFlow model optimization	2
Model understanding	Integrated gradients, Uncertainty quantification with SNGP, Probabilistic regression	
Reinforcement learning	Actor-critic method, TensorFlow agents	2
TF.estimator	Premade estimator, Linear model, Keras model to estimator, Multi-worker training with estimator, Feature columns	5

“Save and load". The beginner’s “Save and load" tutorial covers various methods for saving TensorFlow models during and after training, including checkpoints, entire models, and loading saved models. In contrast, the advanced tutorial for “Save

and load," found under “Distributed training", focuses on the specific logistics related to saving and loading models within distributed training models using distribution strategies. Our scanning through the content of the beginner and advanced tutorials for this subtopic demonstrated extensive differences in the subject matter. The overlapping subtopic is shown in teletype font in Table 2.2.

Analyzing tutorial's readability

To assess the nature of TensorFlow tutorial documentation, we analyzed each tutorial under beginner and advanced competence levels to determine the Flesch readability score, sentence, and word count. Subsequently, we statistically compared these documentations using the Mann-Whitney test and Cohen's effect size.

TABLE 2.3: P-value and effect size results for readability, total sentences, and words, number of examples and lines of code

Figure	Attribute	p-Value	Effect Size
3.(a)	Flesch Ease readability	*0.002**	- 0.676
3.(b)	Total # of sentences	0.823	+ 0.165
3.(c)	Total# of words	0.616	+ 0.073
3.(d)	Number of examples	*0.037**	- 0.730
3.(e)	Lines of code (LOC)	0.400	+ 0.303

We summarized these findings in Table 2.3. Also, Figure 2.3 displays the boxplot distribution of the Flesch readability score, the total number of sentences, words, the example count, and code size for each beginner and advanced tutorial. When comparing the group means using the Mann-Whitney tests, we identified a statistically significant difference in the Flesch readability ease between beginner and advanced tutorials ($p - value = 0.002$). Figure 2.3 shows that the group means of readability score is significantly higher for beginner tutorials when compared with the advanced ones. In other words, the beginner tutorials are significantly easier to read.

It's important to highlight that there are no statistically significant differences in both the total number of sentences ($p - value = 0.823$) and the total number of words ($p - value = 0.616$). This is reaffirmed by the visual representation in Figure 2.3-(b) and Figure 2.3-(c), which shows an overlap between the two notches of boxplots in

each group. It's also worth noting that both the total number of sentences and the total number of words exhibit a small effect size (`< 0.2`). Moreover, sentence count and word count show a positive effect size.

Comparing example count and integrated code size

We compared the beginner and advanced tutorials, incorporating the lines of code (LOC) and example count. Based on the results in Table 2.3, there is a statistically significant difference in the total number of examples between beginner and advanced tutorials with a negative medium-size effect. However, there is no statistically significant difference in lines of code between the two tutorial levels (beginner and advanced), and it has a small effect size. The more detailed analysis of the obtained results shows that even though the advanced tutorials have fewer examples, those examples contain a significantly higher number of lines of code.

> Qualitative analysis shows no substantial overlap in topics between documentation levels, and there are no significant differences in the total number of sentences, words, or lines of code. However, beginner tutorials include notably more examples. Readability scores differ, with beginners scoring 58.21 and advanced users scoring 54.28 on the Flesch Reading Ease scale.

2.5.2 Questions about TensorFlow Documentation (RQ2)

We gathered a set of 405 TensorFlow documentation-related questions from Stack Overflow (Section 2.3.2) and took a systematic approach (Section 2.4) to answer **RQ2**.

The below example[2] shows a user asked question with a direct hyperlink to a particular TensorFlow documentation. The question is with regards to `tf.data: Build TensorFlow input pipelines` tutorial. The developer explains that they need to switch between iterators of different shapes, while they provided examples on TensorFlow data

[2] https://stackoverflow.com/questions/51997426

input pipelines guide [29], which only explains the process when the iterator switches between the same output shapes:

> TensorFlow documentation has examples regarding switching between datasets using `reinitialize_iterator` or `feedable_iterator`, but they all switch between iterators of the same output shape, which is not the case here. How to switch between training & validation sets using `tf.Dataset` & `tf.data.Iterator` in my case?

The annotation process revealed the following categories in the example, as discussed in Section 2.4.2:

What? Describes the issue - *data adaptation, which involves converting raw data into the library's required format.*

Why? Explains the reason for the issue - insufficient information on exceptions and alternatives when *users struggling with instructions in different domains*.

Which? Refers to the documentation - the official TensorFlow Guide for *CORE TensorFlow modules.*

In Table 2.4, we summarized the categories with relevant Stack Overflow question examples.

What? - Nature of the Questions

We adopted the categories for machine learning queries on Stack Overflow, as originally proposed by Islam et al. [70]. The Kappa agreement between the two annotators was 0.77, considered a substantial agreement [100]. *Errors and exceptions* (24.9%), *model creation* (9.1%), and *parameter selection* (8.6%) are most frequently questioned areas about machine learning documentation. Our analysis yielded just one question for each of the *data cleaning*, *model selection*, *robustness*, and *setup* categories.

It is understandable that many turn to Stack Overflow for solving *errors or exceptions*. The predominance of questions related to *model creation* and *parameter selection* among machine learning queries on Stack Overflow can be attributed to the inherently complex

TABLE 2.4: Taxonomy of the nature (what?), reasons (why?) of questions about TensorFlow documentation on Stack Overflow and the documentation that was questioned (which?).

Taxonomy	Description	Cards %
What?		
Error/Exception	All the errors related to either the Shape Mismatch or Type Mismatch go under this category	24.9%
Parameter selection	Choosing appropriate values for the model to perform in the best possible manner	8.64%
Data adaption	Obtaining the input data, reading it, and encoding it according to the specifications	7.16%
Featuring	Decreasing the data dimension into a more focused feature space to get useful information	7.16%
Method selection	Is concerned with problems involved with the usage of APIs for performing validation	6.91%
Prediction accuracy	Is concerned with issues related to model overfitting or underfitting	5.67%
Performance	Is concerned with issues such as long training time or high memory consumption	5.43%
Model load/store	Is concerned with loading the model on a disk and storing them	4.44%
Model visualization	Is concerned with the visualization representation of images	4.19%
Loss function	Issues on calculating the distance between the actual and expected output, such as log loss.	3.45%
Custom (non-ml)	Is the category defined to reference the outlier categories beyond the scope of ML	3.45%
Optimizer	The function that helps with adjusting the weights and the learning rate of the model	2.71%
Model selection	Is concerned with choosing the best-fit model and API version	2.22%
Model conversion	Issues on transitioning between models trained with different libraries.	1.72%
Output interpretation	Is the analysis of the results of the model in order to draw meaningful conclusions	1.23%
Shape mismatch	Occurs when the matrix or tensor shape doesn't match the next layer in a neural network.	0.49%
Data cleaning	Handling data values like null or missing values that could potentially cause problems	0.24%
Robustness	Is concerned with the model's robustness to input dataset changes in reference to noise	0.24%
Setup	All the conditions to perform the experiment, including the dataset, model, and parameters	0.24%

Continued on next page

Table 2.4 - continued from previous page

Why?		
Inadequate examples	Inadequate information on exceptions and alternatives within examples	64.30%
Documentation ambiguity	Unclear or vague information in a document	28.00%
Documentation completeness	Insufficient documentation on particular settings or configurations	4.19%
Not ML Documentation	Requesting documentation irrelevant to machine learning tool or library	1.48%
Lack of alternative solutions	Requiring a different solution to a problem	0.74%
Document & code discrepancies	The difference between documentation provided and code implementation or syntax	0.50%
Documentation replicability	Problems with replicating the results or examples provided in documentation	0.25%
Request (additional) documentation	Lack of access to documentation due to deprecated links	0.25%
Which?		
TensorFlow tutorials	The official tutorials officially hosted on TensorFlow domain	63.0%
Third-party documentation	Non-TensorFlow owned documentation (e.g., scientific articles, blogs, reports, and websites).	11.3%
TensorFlow guides	TensorFlow hosted documentation are Jupyter notebooks run in Google Colab.	6.3%
TensorFlow GitHub documentation	The Documentation of TensorFlow provided on the project's open source GitHub repository	5.6%
Scientific publications	Written works that report the results of scientific research	2.3%
Multimedia tutorials	Including video tutorials on YouTube or online teaching platforms like Coursera	0.7%

and iterative nature of these processes. *Model creation* involves designing and developing algorithms that can learn from data, a task that requires a deep understanding of various machine learning techniques, data structures, and often a significant amount of experimentation to achieve optimal performance. Similarly, *parameter selection*, which involves tuning hyperparameters to improve model accuracy and efficiency, is a critical yet challenging step in the machine learning pipeline. Hyperparameters can greatly influence the behavior and success of a model, but finding the right combination often requires extensive trial and error and specialized knowledge. As a result, practitioners frequently seek guidance and insights from the community to navigate these intricate

and technically demanding aspects of machine learning, leading to a higher volume of related questions on platforms like Stack Overflow.

Why? - Motivation for the questions

As a result of this process, one annotator identified 12 overall categories, while the other identified seven categories. The two annotators discussed the categories, merged the five common categories identified, and rephrased the category names. With the help of a mediator (one of the authors of this paper), they discussed the remaining differences and identified three more categories, then re-evaluated the questions for the task. As a result of this, we ended up with a set of eight categories to answer the “why" question (See Section 2.4.2 for protocol details).

According to our findings, the documentation for machine learning is primarily subject to questions when developers apply a documented process or example in different application domains, accounting for 64.3% of the cases. In the second place, 28.0% of the questions are related to *document ambiguity* in the TensorFlow libraries. *Document completeness and mistakes* also triggered 4.2% of the questions. *Documentation including bugs and inconsistencies* (0.5%), *searching for alternative documentations* (0.7%), *replicating and taking the steps explained in a documentation* (0.2%), and *requesting and accessing other documentations* (0.2%) were also among the identified triggers. We also categorized 1.5% of the questions as unrelated to the documentation itself, and the root cause of the question was in programming issues while the users were referring to the documentation as well.

Figure 2.4 demonstrates the heatmap on the intersection of the what (nature of the questions) and why (the trigger of the questions). Overall, the “errors and exceptions" caused by “inadequate examples” have the highest proportion of 16.3% of the issues related to TensorFlow documentation. Secondly, 7.1% of the documentation questions are due to “errors and exceptions" occurring because of “documentation ambiguity". Additionally, 5.4% of the questions (22 questions) were raised during “modal creation"

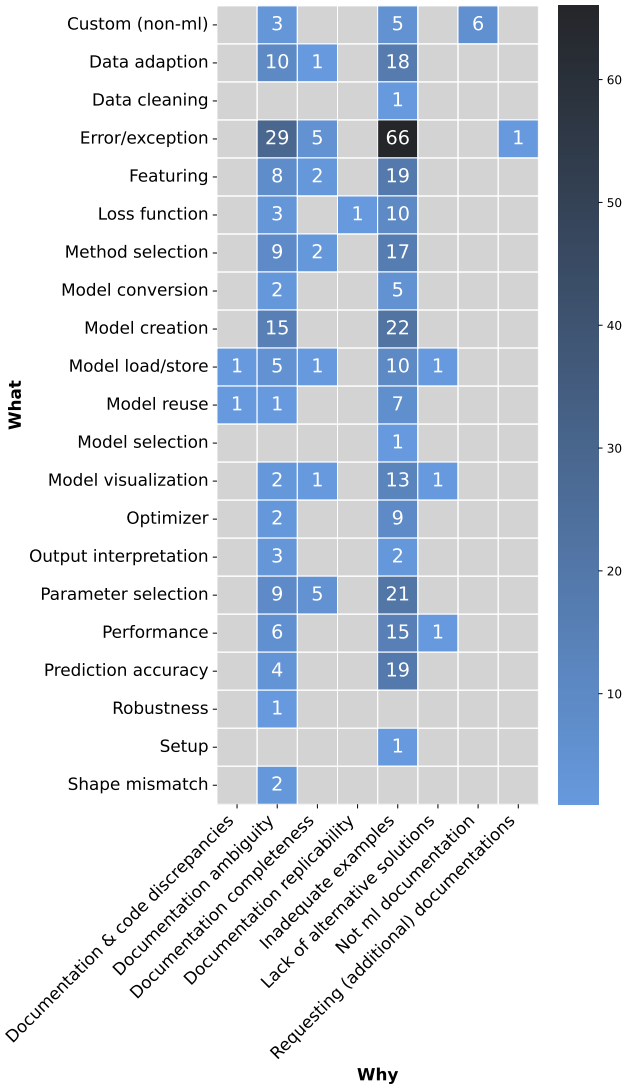


FIGURE 2.4: Frequency of all questions on What area vs Why for Stack Overflow Questions about TensorFlow.

due to “inadequate examples", and 5.1% of the questions were raised about “parameter selection" again due to “inadequate examples".

Which? - The documentation subject to questions

We also discussed the type of documentation used and referenced in the Stack Overflow questions. According to the data obtained incorporating an open card sorting (see

Section 2.4.2), 97% of the questions we identified are related to TensorFlow documentation (API documentation, tutorial, or guide). Among them, the majority of the questions were regarding TensorFlow tutorials (66.9%). Moreover, the rest of the questions are subjected to TensorFlow guides (6.3%), TensorFlow GitHub documentation (6.8%), third-party documentation (11.3%), scientific publication (2.0%), and multimedia tutorials (0.5%). These observations revealed that the majority of the users face problems when using *official* TensorFlow tutorials or other official documentation.

Almost a quarter (24.9%) of questions regarding TensorFlow documentation were related to errors and exceptions, which are attributed to insufficient or unclear examples. Also, a significant majority (64.3%) of the questions were due to the inadequacy of the provided examples and details about alternative solutions. These questions are mostly about the TensorFlow tutorials.

2.5.3 Comparing Users Who Pose Questions about Beginner and Advanced Tutorials (RQ3)

To address **RQ3**, we analyzed Stack Overflow user profiles to determine if there are differences in the background knowledge of users who asked questions about beginner tutorials compared to those who asked about advanced tutorials. Specifically, we compared overall user reputations, v-index in Python, reputation in Python, months of experience in Python, v-index in TensorFlow, reputation in TensorFlow, and months of experience in TensorFlow. We performed this analysis for users asking questions about each tutorial on Stack Overflow. Figure 2.5 shows the comparisons.

When comparing the Stack Overflow reputations assigned to users asking about beginner-level tutorials versus those inquiring about advanced-level tutorials, we found no significant difference. The analysis yielded a p-value of 0.121 and an effect size of 0.077. This finding remained insignificant when comparing their v-index in Python or TensorFlow. Using the Mann-Whitney test to compare the v-index between the two

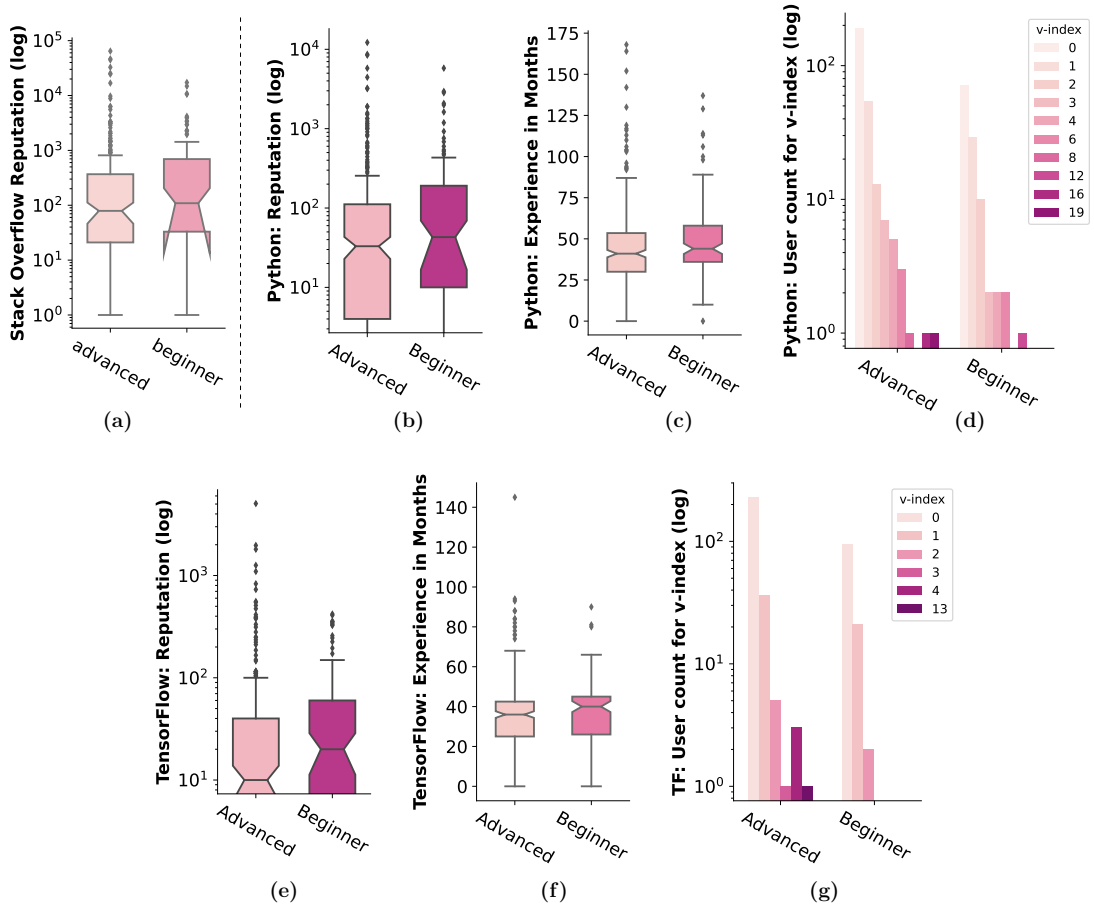


FIGURE 2.5: **(a)** Users Stack Overflow reputation **(b)** Python-specific calculated reputation **(c)** Python-specific experience in months **(d)** Python-specific v-index **(e)** TensorFlow-specific calculated reputation **(f)** TensorFlow-specific experience in months and **(g)** TensorFlow-specific v-index.

groups, we obtained a p-value of 0.198 with an effect size of -0.047 for Python, and a p-value of 0.7 with an effect size of 0.064 for TensorFlow, both of which were insignificant.

Further, we computed the Python and TensorFlow-specific reputations using the attributes listed in Section 2.4.3. This process was carried out separately for questions associated with Python or TensorFlow hashtags. However, we did not observe any significant difference in the reputation of users asking questions about beginner tutorials versus advanced tutorials in Python or TensorFlow. This is supported by the p-values of 0.06 and 0.460, and the relatively weak effect sizes of 0.035 and 0.107 for Python and TensorFlow, respectively.

Conversely, when comparing the months of experience in Python and TensorFlow

TABLE 2.5: Mann Whitney test p-value and Cohen's effect size

Figure	Attribute	p-Value	Effect Size
Python			
5.(b)	Reputation	0.060	+ 0.035
5.(c)	Experience in months	0.036	- 0.146
5.(d)	v-index	0.198	- 0.047
TensorFlow			
5.(e)	Reputation	0.460	+ 0.107
5.(f)	Experience in months	0.126	- 0.010
5.(g)	v-index	0.700	+ 0.064

based on the time elapsed since each developer's first question with these tags, we found a significant p-value of 0.036 for Python. This indicates a notable difference in the distribution of months of experience in Python between users asking about beginner tutorials and those asking about advanced tutorials.

Notably, developers asking beginner-level questions have more months of experience in Python. However, it's important to note that this p-value is accompanied by a relatively weak negative effect size (-0.146), leading us to consider this finding inconclusive. Furthermore, we obtained a p-value of 0.126 for months of experience in TensorFlow, indicating no significant difference, along with an effect size of -0.01. Table 2.5 and Figure 2.5 summarize these comparisons.

We observed no significant difference in developers' profiles when posing questions about beginner or advanced tutorials, considering overall reputation, Python or TensorFlow-specific reputation, or v-index

2.6 Threats to Validity

There are multiple threats to the validity of our case study. In particular, the measurements we used could impose some *construct validity* on our results. We presented results of the Flesch Reading Ease [81]. However, there are a number of different metrics with slightly different algorithms to measure readability. We evaluated and compared the

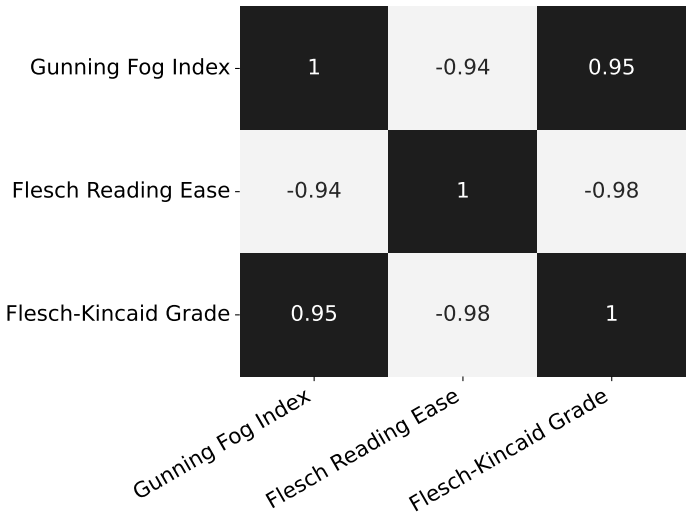


FIGURE 2.6: Correlation matrix of the readability scores

readability of beginner and advanced tutorials using three Gunning Fog Index, Flesch Reading Ease, and Flesch-Kincaid Grade metrics and found a high correlation between these values (min of 0.94 correlation and average of 0.957). Hence, we relied on the most popular metric, Flesch Reading Ease. Figure 2.6 illustrates the correlation matrix between three readability metrics.

As for the *conclusion validity*, we used the reputation, v-index, and months of experience in the analysis of **RQ3** while we drove on top of the latest state-of-the-art methods [178] still there is a chance that the user profiles might have differences in other metrics not studies in this paper. However, we believe that the overall look into state-of-the-art measures from Stack Overflow and even further separating them based on the Python and TensorFlow skill set is reducing such risk. Using months of experience as a metric may pose a threat to validity. Experienced users without a Stack Overflow account could create new accounts and ask advanced questions, and users might no longer work in the same domain, potentially providing misleading information about their experience. However, the diverse range of data points and users across experience levels in our study helps mitigate this impact on our results. Additionally, we used the non-parametric Mann-Whitney U test in this study. Our reliance on a single test for multiple

comparisons raises the possibility of statistical errors. We mitigated this threat by adjusting the thresholds using Bonferroni correction.

As for the *external validity*, the case study is carried out by choosing Python as the software engineering domain. For instance, if the domain is changed from Python to Java, the observation might change. However, that deviation won't be significant since we mitigate that problem by deriving attributes based on the Stack Overflow user profile, which is the typical feature for all users in all the domains. The formation of two taxonomies is prone to *external validity*. Establishing "fully correct" taxonomies is inherently challenging. We used established card sorting techniques to this end [195] and leveraged the strengths of both open and closed card sorting (**RQ2**). The high Fleiss' Kappa value indicated agreement among annotators. As for the *internal validity*, we hypothesized that the skills of a particular user are independent of each other. However, in the real world, the skills are not independent. This may impose some internal validity in our modeling while we believe the chances are low as this was not further investigated in the relevant studies on developers' experience analysis either [172, 91].

2.7 Implications and The Potential Futures

Currently, the techniques for the automated generation of software documentation, code comments, release notes, etc. (Table 2.6), are outputting a single document for all users. Researchers can further use our results to curate these automatically generated documents to match the users' skills and level of proficiency.

TensorFlow is the only machine learning library that tailors part of its documentation for different competency levels. We aimed to determine if these documents are well-suited for its users and explored it in three ways: (i) We compared the topics and content of beginner versus advanced tutorials. We found only one overlapping topic, but the content was entirely different (**RQ1**). (ii) We analyzed the nature and root cause of questions related to both beginner and advanced tutorials. There was no significant

difference in the type or cause of questions between the two levels (**RQ2**). (iii) We examined developer questions about TensorFlow tutorials by looking at overall Stack Overflow reputation, months of experience in Python and TensorFlow, v-index in each skill, and skill-specific reputation. We found no statistically significant differences, suggesting that developers with a variety of backgrounds and knowledge levels asked questions about both beginner and advanced tutorials (**RQ3**).

Our results indicate that although TensorFlow attempts to differentiate developers' proficiency levels, the distinction remains ambiguous. Furthermore, our findings underscore the importance of personalizing documentation to align with users' skill levels. Overlooking this issue can lead to confusion, an increase in user inquiries, and a surge in third-party tutorials and documentation. However, with the advent of generative models, we now have unprecedented opportunities to customize documentation and software artifacts more effectively.

The current literature on automatic documentation generation and expertise analysis has largely treated these fields as distinct, without considering the users' expertise in the generated documentation (refer to Table 2.6). In addressing this gap, it is crucial to integrate user expertise for a given developer with the skills required to comprehend a document. Below, first we provide a formal model of this problem. Then, we elaborate on the positioning of the future trend considering advances in generative AI.

2.7.1 Modeling Customization in Software Documentation

Let's consider a set of N developers $\mathcal{D}$ where each developer possesses skills where $\mathcal{S}$ is a finite countable set of M skills. For any given skill, different developers may have different levels of proficiency; as in real life, proficiency levels are challenging to measure and are often quantified in a few categories. For simplicity, we assume that the granularity of these skills is similar and quantified into the same number of levels. Each developer $d \in \mathcal{D}$ has a set of s skills $Skills(d) = \{skl_1, .., skl_s\}$ helping them to excel in their career. This is to say, each skill $s_k \in \mathcal{S}$ is modeled by means of L skill levels: $\mathcal{L} = \{lvl_1, lvl_2, ..., lvl_L\}$. Levels are ordered according to a compatibility

relation $\leq$, i.e., if the developer d_i and their skill sk_l has level lvl_r and developer d_j for the same skill has level lvl_q with $lvl_q \leq lvl_r$ then d_i has a higher competence. As a first-order approximation, we also assume all the skills are independent of each other: $\forall sk_i, sk_j \in \mathcal{S} | sk_i \neq sk_j : sk_i \perp sk_j$. Each developer must have at least one skill.

$$\forall d \in \mathcal{D} : |Skills(d)| \geq 1 \tag{2.1}$$

Different developers may have different levels of different skills; for instance, a developer might be quite a novice in mastering a programming language like Python and be advanced in a machine learning library such as TensorFlow, while another may be quite advanced in both machine learning and Python. In this example $L = 2$ (i.e., $\mathcal{L} = \{$ novice, advanced in programming with Python $\}$). Given that the developer d has a set of $Skills(d)$, they have also associated a level for each of their skill. In other words:

$$\forall d \in \mathcal{D}, \forall sk \in Skills(d) : Level(d, sk) \in \mathcal{L} \tag{2.2}$$

At the same time, each code snippet is related to a set of O documentation, $Documentation = \{doc_1, doc_2, ..., doc_o\}$. To be understood, document doc_k requires certain skills and skill levels, i.e., the reader should have *compatible* skills and skills levels. This is to say, if a document o requires skills $Skill(o)$ with levels: $Levels(o, sk)$ to understand it, the developer d must have skills such that:

$$\forall sk \in Skills(o) : sk \in Skill(d) \,\&\, Level(o, sk) \leq Level(d, sk) \tag{2.3}$$

Otherwise, the developer raises a set of questions, $Question = \{Qst_1, Qst_2, ..., Qst_q\}$ where developer d asking a question q about code documentation o. We attributed the ambiguities in the documentation (on terminology, etc.) which may trigger questions, to the developers' competency for simplicity.

2.7.2 The Future: Machine Learning for Automated Software Documentation

The integration of machine learning into software development is rapidly transitioning from visionary concepts to actionable implementations, particularly within integrated and context-aware development environments. The current state of 'copilots' and similar tools is a testament to this, marking a significant step towards realizing a more intuitive and integrated development process [17, 119]. These advancements reflect a broader exploration within computer science—not limited to software engineering—aimed at understanding user proficiency through diverse data points, including developers' queries, commit histories, and organizational tenure.

Vision for Enhanced Documentation through Generative AI Building on these foundations, we foresee a transformative future where generative models significantly tailor software documentation to the individual needs of developers across various disciplines. This bespoke adaptation extends beyond traditional documentation formats like tutorials and API documents to include multimedia elements and interactive components that evolve in real-time based on user interaction and feedback and the content created on multimedia platforms such as YouTube. Such dynamic documentation strategies are poised to minimize confusion and streamline the developer experience significantly.

By leveraging generative AI, documentation can be tailored to meet the specific needs of individual users, providing a more personalized and effective learning experience. Additionally, large language models can analyze user queries, understand context, and generate contextually relevant explanations. This can be particularly beneficial in crafting documentation that not only aligns with the users' proficiency but also anticipates their potential questions or challenges. This practical implications is facilitated by LLMs and models like GPT [189, 38, 95].

TABLE 2.6: Studies on generating software documentation and whether they consider developers' skills in their design.

Generated Document	Reference	Developer skills?
Release Notes	Ali et al. [8, 9]	✗
	Nath and Roy [122]	✗
	Moreno et al. [115, 117]	✗
	Jiang et al. [73]	✗
	Kamezawa et al. [78]	✗
Code Comment	Hu et al. [67]	✗
	Wang et al. [176]	✗
	Jiang et al. [74]	✗
	Liu et al. [97]	✗
	Jia et al. [72]	✗
	Mu et al. [118]	✗
Commit Messages	Jiang et al. [75]	✗
	Liu et al. [94]	✗
	He et al. [63]	✗
	Vu et al. [174]	✗
Pull Request Description	Liu et al. [96]	✗
	Isran et al. [69]	✗
	Fang et al. [50]	✗
API documentation	Treude et al. [165]	✗
	Rocha et al. [141]	✗
	Guerrouj et al. [57]	✗
	Robillard et al. [139]	✗
	Nybom et al. [124]	✗

2.8 Related Work

In the field of machine learning, documentation is frequently discussed, with an emphasis on reporting proper information alongside the training and testing data and code to reduce what is termed documentation debt [20]. Various efforts have aimed to improve machine learning documentation, primarily by better empirically explaining the data [20, 144]. Particularly, Chang et al. [35] surveyed 24 AI/ML practitioners, reflecting on the practices of implementing and operationalizing ML documentation in their workflows, identifying challenges, and addressing them. Conversely, there is a well-established body of research focused on software documentation. With the advent of

machine learning, the software aspects of machine learning libraries have become the subject of numerous studies. This section provides an overview of the most relevant work related to our study.

2.8.1 Software Documentation and its challenges

Using Stack Overflow to identify developers' questions and pain points has been an established study design in software engineering [7, 109]. In particular, it was used to identify issues of software documentation [5, 165, 19, 128]. Aghajani et al. [5] performed an empirical study using mailing lists, Stack Overflow, and project repositories to identify issues developers faced using software documentation. They further performed a survey to evaluate the mined repository results against developers' perspectives [6]. Further, Treude et al. [164] suggested a framework to evaluate the quality of software documentation. Venigalla et al. [173] underscored documentation challenges by mining developers' sentiment in commit messages, which revealed that 45% expressed a "trust" emotion.

Software documentation, meeting developers' information needs passively, has prompted the development of automated tools and techniques for generating and enhancing documentation based on code and developers' activities [44, 107, 140]. An overview of these studies and the targeted document they generate is provided in Table 2.6. In particular, this table shows that none of these techniques considers the users' skill level and hence only develops one documentation, hoping that it fits every user.

2.8.2 Software Engineering for Machine Learning

MLOps, or Machine Learning Operations, is a set of practices that combines machine learning, DevOps (development and operations), and data engineering to automate and streamline the deployment and management of machine learning models in production [84, 76]. Along with the MLOps, different levels and granularity of documentation have also been discussed. MLOps emphasize on documentation needed for each step

of software life cycle [194]. Particularly the documentation of functional and non functional requirements for machine learning software has been discussed [18, 23]. Warnett et al. [180] investigated the understandability of MLOps system architecture descriptions and found that semi-formal MLOps system diagrams improved comprehension. Furthermore, the iterative nature of ML application development, with its variety of artifacts, makes achieving comparability, traceability, and reproducibility challenging. Schlegel et al. [149] conducted a literature review on ML artifact management systems (AMS), identifying current AMSs and their functional and non-functional properties and the role of documentation.

Software documentation has traditionally been written or automatically generated for software developers. However, there has been no specific study on machine learning documentation nor any that considers the matching of content with multi-faceted users' competence.

2.9 Conclusion

Traditionally, software documentation has been developed by software developers for software developers. However, the use of machine learning libraries by domain experts who are not necessarily professionals in software engineering is increasing. Along with the discussions and applications of *software engineering for machine learning*, we explored the match between the skill level of software developers and the documentation by conducting a case study on TensorFlow, a widely adopted library and tool set. We mined Stack Overflow questions related to these documentations, constructing a taxonomy of issues encountered by developers. We comprehensively compared beginner and advanced tutorials, evaluating their content in terms of readability and technicality. Furthermore, we analyzed the profiles of users who posed questions on each tutorial. Our findings showed that TensorFlow distinctly segregates topics for beginner and advanced tutorials, with no advanced discussions on the same subjects. Interestingly, all tutorials attract questions from developers with varying skill levels. Notably, beginner

tutorials receive extensive inquiries from well-seasoned developers. We argue that as machine learning software becomes more widely used across domains, its documentation should be tailored to accommodate a variety of skills.

Chapter 3

Retrieval-Augmented Editing of TensorFlow API Documentations based on Stack Overflow Questions

Abstract

With the rapid growth of machine learning libraries, accurate and comprehensive API documentation has become essential for guiding developers in effectively using these tools and services. Modern software development frequently involves updates—major, minor, and patches—to enhance stability and performance. However, documentation often lags behind these updates, resulting in increased documentation-related issues on GitHub and questions on Stack Overflow. Moreover, further analysis on the Stack Overflow questions related to TensorFlow API documentation revealed that the majority of these questions arise from inadequate examples, documentation ambiguity, and requests for additional resources. To address this challenge, we propose DocChameleon to automatically augment TensorFlow API documentation by addressing documentation-related questions from Stack Overflow, using Large Language Models (LLMs) and the Retrieval-Augmented Generation (RAG) technique. DocChameleon generates executable code examples to tackle the lack of examples, clarifies ambiguities

with detailed explanations, and provides related YouTube tutorials and Stack Overflow posts to acquire relevant API knowledge.

We conducted a comprehensive quantitative study to assess the inclusion of relevant information in the data retrieved by DOCCHAMELEON for addressing questions, and compared this with state-of-the-art techniques. Additionally, we evaluated the correctness, similarity, and faithfulness of the explanations and code examples generated by DOCCHAMELEON, as well as the executability of the generated code examples. We also conducted a qualitative study to evaluate developers' perspectives on the TensorFlow API documentation augmented by DOCCHAMELEON. Our results indicate that DOCCHAMELEON retrieves information from Stack Overflow that contains 30.35% more relevant content for addressing questions compared to the baseline methods. We demonstrated that the explanations and code examples generated by DOCCHAMELEON show better correctness, similarity, and faithfulness with respect to the ground truth answers. Moreover, DOCCHAMELEON generated executable code examples addressing TensorFlow documentation-related questions with an execution success rate of 56.25%. Finally, the user study revealed that 93.24% of the participants prefer using DOCCHAMELEON augmented TensorFlow API documentation for programming tasks.

3.1 Introduction

Software applications are continuously broadening their scope and are now used across a wide range of fields, including life sciences, economics, the fashion industry, and manufacturing. While developers are typically familiar with their specific application domains, their levels of education and experience in programming and software engineering can vary widely [143]. Traditionally, software developers create documentation for their code, primarily focusing other developers within the same field. In modern software development, developers are also relying more on API libraries to speed up the development process [130]. API documentation is crucial for enhancing the usability of APIs [110] and addressing their usability issues [120]. However, the API documentation of modern libraries and frameworks often suffers from incomplete or outdated content,

unclear explanations, and insufficient examples [170, 4, 164]. Automatic API documentation augmentation has been studied in software engineering literature [121, 190, 169, 80]. These approaches involve using code examples and reviews from technical Q&A sites [168], extracting "insight sentences" from Stack Overflow [166], and analyzing software change history [15] to automatically improve the quality of API documentation over time. However, these studies are often conducted without considering the underlying causes of the documentation-related issues that developers face.

With the emergence of machine learning, there is an increasing trend of machine learning and Python-related questions on Stack Overflow [11, 185]. As machine learning continues to gain popularity, individuals from diverse fields are integrating it into their workflows through programming. Poor documentation of machine learning components can reduce their usefulness, delay time to market, and restrict adoption [61]. Existing research has addressed "documentation debt" in training datasets [24] and included systematic studies on machine learning model documentation. However, no specific research has focused on machine learning API documentation [25]. Despite the growing popularity of machine learning, there remains a gap in the literature concerning the generation and augmentation of machine learning documentation. Therefore, we explored the potential of leveraging generative AI for augmenting machine learning API documentation to address machine learning API documentation-related issues on Stack Overflow.

Large Language Models (LLMs) have demonstrated exceptional performance in a range of natural language processing (NLP) tasks [192] by leveraging their strengths in reasoning, comprehension, question answering, and code generation [85] . Recent advancements in LLMs, including increased model sizes and larger training datasets, have significantly enhanced their performance in many NLP applications [82]. However, LLMs might generate responses with factual inaccuracies, or "hallucinations" [89, 33, 135, 71], due to biases in their training data and lack of access to up-to-date information [62], proprietary, or domain-specific knowledge [150]. Retrieval Augmented Generation (RAG) helps to mitigate this issue by enhancing model's generation capacity for

high factual accuracy with relevant contextual information [88, 87]. RAG consists of two core components: (i) *retrieval* of relevant contextual information from various available data sources and (ii) *generation* using the information obtained during retrieval [54].

In this study, we propose DOCCHAMELEON for automatic machine learning API documentation augmentation, to enhance TensorFlow API documentation. DOCCHAMELEON takes Stack Overflow questions related to an API documentation as input and addresses them using LLMs and the RAG technique. Our in-depth analysis of TensorFlow documentation-related questions on Stack Overflow, which focused on identifying the nature and causes of these questions and building a taxonomy (see Section 3.2), revealed that 92.55% of these questions result from ambiguities in the documentation, insufficient code examples, and requests for additional resources to gain the background knowledge needed to resolve API issues. Therefore, DOCCHAMELEON augments API documentation by including (1) generated explanations, addressing documentation ambiguities as users often request explanations to resolve those ambiguities, (2) generated code examples, since a majority of documentation-related issues arise due to the lack of code examples needed to clarify undocumented aspects of the API, (3) links to relevant YouTube tutorials, as developers frequently turn to YouTube for gaining API knowledge [47], and (4) links to related Stack Overflow Q&A posts, as Stack Overflow is widely recognized as a valuable resource for learning APIs by offering API knowledge through a combination of textual descriptions and code examples. Additionally, these resources address various undocumented issues that developers encounter in practice, along with their solutions [186].

To evaluate the performance of DOCCHAMELEON, we first conducted a quantitative analysis comparing the relevancy of the information retrieved by DOCCHAMELEON for addressing documentation-related questions against three state-of-the-art techniques. We also examined the correctness, similarity, and faithfulness of the explanations and code examples generated by DOCCHAMELEON based on ground truth answers. Next, we evaluated the executability of the TensorFlow code examples generated by DOCCHAMELEON. Additionally, we conducted a qualitative analysis to assess developers'

perspectives on the information added by DOCCHAMELEON to the TensorFlow API documentation. Based on our quantitative and qualitative results, we argue that addressing documentation-related questions to augment machine learning API documentation can provide usage information not covered in the official API documentation, which enhances users' understanding of the API. Additionally, generative AI can be effectively leveraged to automate this documentation augmentation process, enabling dynamic documentation.

Our core contributions from this research are as follows:

- We propose a tool called DOCCHAMELEON to augment TensorFlow API documentation based on the Stack Overflow questions that are directly relevant to the API documentation.
- We conducted a qualitative analysis to assess the relevancy of information retrieved by DOCCHAMELEON for addressing documentation-related questions, comparing it against three baselines. We also evaluated the correctness, similarity, and faithfulness of the explanations and code examples against ground truth answers, as well as the executability of the TensorFlow code examples generated by DOCCHAMELEON.
- We assessed the relevancy, helpfulness, and accuracy of the explanations, code examples, YouTube tutorials, and Stack Overflow post links included by DOCCHAMELEON in the TensorFlow API documentation through a user study.

The rest of the paper is organized as follows. In Section 3.2, we presents the motivation of this study. Section 3.3 outlines the related works and Section 3.4 details the empirical data used. In Section 3.5, we describes the implementation details of DOCCHAMELEON. Section 3.6 reports the evaluation setup. Subsequently, the implications and the future works are explored in Section 3.8. Section 3.7 presents the results. Potential threats to validity are addresses in Section 3.9. Finally Section 3.10 concludes the paper.

3.2 Motivation and Background

Various studies highlight the increasing trend of machine learning and Python-related questions on Stack Overflow [184, 10]. Among *Caffe*, *Keras*, *Mahout*, *MLlib*, *Scikit-learn*, *Theano*, and *Torch* machine learning libraries, TensorFlow is recognized as the most used machine learning library [162] and it has the highest number of highly-rated posts on Stack Overflow [70].

Thus, we explored the challenges developers face with TensorFlow documentation by analyzing questions on Stack Overflow. Using the Stack Exchange Data Explorer, we gathered 45,866 questions tagged with both TensorFlow and Python. From this dataset, we randomly sampled 1,000 questions. Two annotators, each with over a year of professional software engineering experience and holding M.Sc. degrees in software engineering, classified these questions into "documentation-related" and "non-documentation-related" categories, identifying 124 documentation-related questions. Subsequently, we removed code blocks using HTML tags and employed `NLTK` to identify Part-Of-Speech (POS) tags, focusing on nouns, adjectives, and verbs. The words were then lemmatized, and their TF-IDF values were computed to rank their frequency in both documentation-related and non-documentation-related questions. We excluded common words between the two lists, identifying keywords specifically associated with documentation questions (*documentation*, *document*, *doc*, *website*, *official*).

Using these keywords, we identified an additional 6,901 questions from the remaining 44,866. We randomly selected 2,000 of these questions, and the annotators again manually inspected them, identifying 281 as documentation-related. With a total of 405 documentation-related questions, the annotators analyzed their nature and underlying causes. They employed closed card sorting [195] to classify the questions into 27 predefined categories by Islam et al. [70], followed by open card sorting [195] to group them based on the similarity of their causes. The annotators then discussed and agreed upon category names that best represented the issues. Finally, a moderator resolved any disagreements, resulting in a comprehensive taxonomy of issues encountered by developers when using TensorFlow documentation. According to our taxonomy, 28.0%

of the questions related to documentation ambiguity, where users request explanation to solve the ambiguity. Furthermore, 64.3% of the questions were related to inadequate examples. Additionally, 0.25% of the questions requested additional resources for understanding API usage.

Our further in-depth analysis of Stack Overflow tags related to TensorFlow reveals that Python is the most frequently used tag. Its usage is 1.9 times higher than the second most popular tag, "Keras", and 48.7% higher than that of Java. Thus, we examined the timelines of TensorFlow releases (major, minor, and patch), including TensorFlow Python API documentation-related issues on GitHub and questions on Stack Overflow. We observed 102 releases (21 major, 68 minor, and 13 patches) from January 2016 to July 2023. Additionally, we retrieved GitHub issues raised related to TensorFlow API documentation by filtering issues with the `type:docs-bug` label. During the same period, we observed 877 documentation-related GitHub issues and 507 Stack Overflow questions related to TensorFlow Python API documentation (see Section 3.4.1). We analyzed the impact of TensorFlow releases (both major and minor) on GitHub issues and Stack Overflow questions related to TensorFlow API documentation by examining the variation in the number of issues and questions one month before and one month after each release. The results indicated a 15.61% increase in GitHub issues related to TensorFlow API documentation following a major release, accompanied by a 13.90% increase in related Stack Overflow questions. In contrast, no significant variation was observed in the number of GitHub issues or Stack Overflow questions before and after a minor release. Figure 3.1 illustrates a snapshot of the timeline of releases, issues, and questions related to TensorFlow API documentation from January 2021 to July 2023. It also shows a trend of increased documentation-related issues on GitHub and questions on Stack Overflow following major release.

The analysis of the underlying causes of TensorFlow documentation-related questions on Stack Overflow reveals that addressing issues related to inadequate examples, documentation ambiguities, and requests for additional resources resolves 92.55% of

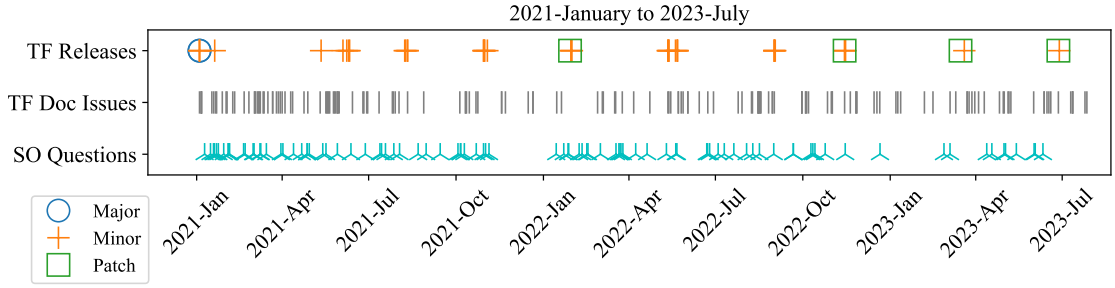


FIGURE 3.1: Timeline of TensorFlow releases, documentation-related GitHub issues and Stack Overflow questions

these questions. Furthermore, the increasing trend of TensorFlow API documentation-related issues on GitHub and questions on Stack Overflow following major releases highlights that the API documentation does not keep pace with TensorFlow releases. Providing explanations and code examples that directly address documentation-related questions through generation will help resolve undocumented aspects of the API, thereby reducing the growing number of such questions following major releases. Furthermore, since users request additional resources to understand the API and resolve these questions, offering relevant YouTube tutorials — widely used by developers for API learning [47] — and Stack Overflow posts — a highly regarded resource that provides textual descriptions and code examples for API learning [186] — will equip developers with the necessary background knowledge related to the API. Motivated by these findings, we argue that an automated tool is needed to enhance TensorFlow API documentation after major releases by providing examples, clarifying ambiguities, and offering additional resources based on documentation-related questions.

3.3 Related Works

The existing literature extensively discusses the automatic generation and augmentation of software documentation. Additionally, several studies have aimed on machine learning documentation.

3.3.1 Automated documentation generation

Several researchers have contributed towards the generation of documentation by proposing various techniques and tools. Raymond et al. [31] generated documentation for exceptions using the conditions that cause the exceptions using symbolic execution and inter-procedural dataflow analysis. DELTADOC is an algorithm designed to summarize the runtime conditions necessary for control flow, documenting program changes to reduce human effort and adding more log information compared to existing version control log messages [32].

Several studies have focused on generating summaries using Java methods. Additionally, there have been efforts to establish traceability links to connect terms in the documentation with source code elements [34] and to generate recommendations on which code elements should be documented [42]. Initially, Sridhara et al. [153] used method signatures and bodies to generate descriptive summary comments for Java methods, which include selecting important or central code statements and expressing that content using text generation. Later, McBurney et al. [108] introduced an improved technique to leverage the context surrounding a Java method, obtaining information about dependencies and other methods that rely on the corresponding method. Additionally, Guerrouj et al. [57] utilized Stack Overflow posts to summarize the use and purpose of methods or classes by analyzing the context surrounding the code elements, allowing developers to quickly understand library identifiers discussed in the documentation.

For release note generation, source code change summarization [117, 114] has been widely adopted, and a recent study conducted by Ali et al. [9] introduced an automatic release notes generation technique. This method involves extracting changes between the previous and new releases, summarizing all relevant information, extracting issues from the issue tracker, and generating the release notes. Several studies have utilized Stack Overflow as a source for automatically generating summaries and API documentation using code examples. Naghshzan et al. [121] introduced a technique for automatic summary generation from Stack Overflow by leveraging topic modeling with

BERT embeddings and the c-TF-IDF technique. Uddin et al. [169] proposed two documentation algorithms: one that visualizes usage and review statistics of API code examples, and another that clusters conceptually similar usage scenarios by mining code examples and reviews from Stack Overflow. Despite efforts to make API documentation more understandable and detailed using examples, their complexity has not been taken into account. Thus, Rocha et al. [142] generated tutorials for APIs using Stack Overflow content, considering their complexity.

3.3.2 Automated API documentation augmentation

Numerous studies have focused on augmenting or improving documentation. Stylos et al. [154] introduced Jaedit that leverages other programmers' previous API usage information for common tasks. It allows developers to augment API documentation by adding usage information using placeholders. Chen et al. [36] automatically integrated crowdsourced frequently asked questions into API documentation by capturing developers' document reading and web searching behaviors. CoDocent utilizes code search engines to link code examples with API documentation. It provides class cloud views and API usage tip diagrams to guide developers in finding code examples while summarizing the outline of a code example [187]. Hoffman et al. [65] augmented API documentation with executable test cases and their expected outputs using Roast to write automated test cases for C, C++, Ada, and Java programming languages. Montandon et al. [113] introduced APIMiner, a tool that enhances Java-based API documentation with code examples extracted from curated source code repositories. APIMiner extracts, summarizes, and ranks code examples, storing them in a database for later retrieval to augment API documentation based on the relevant API methods.

Utilizing social media platforms such as blogs, developer forums, and Q&A sites to augment documentation has been studied by Parnin et al. [127]. They performed web searches for API methods and analyzed the information sources available to developer. Stack Overflow, in particular, has been extensively used as a knowledge source in numerous studies. Subramanian et al. [155] introduced Baker, a tool that supports

both Java and JavaScript, which links source code examples from Stack Overflow and GitHub Gist to API documentation. Baker employs a constraint-based technique to identify fine-grained type references, method calls, and field references in source code snippets. eXoaDocs is another method that enhances API documentation by integrating code examples. It first builds a repository of candidate code examples using the Google code search engine, then summarizes these examples and finally embeds those to the API documentation [80]. However, it lacks a mechanism to ensure the quality of the code examples and to map code samples to correct usage scenarios. To address this issue, Zhang et al. [190] introduced ADECK, a tool that leverages Stack Overflow Q&A to identify and link code examples to API documentation. ADECK extracts the corresponding usage scenarios and subsequently augments the API documentation. Treude et al. [166] introduced SISE, a machine learning-based technique that extracts useful information from Stack Overflow by performing text similarity analysis.

3.3.3 Machine Learning Documentation

Research on machine learning dataset documentation has revealed that training data often suffers from "documentation debt" [24]. To address this issue, Bandy et al. [20] provided comprehensive documentation for the widely-used machine learning dataset, BookCorpus, by applying a datasheet framework. This approach retrospectively documented the dataset's motivation, composition, and collection proces. Moreover, Chang et al. [35] thoroughly explored the implementation and operationalization of machine learning documentation by identifying and addressing the associated challenges through insights gathered from organizations active in documentation practices across various implementation stages. In addition, Bhat et al. [25] conducted a systematic study on machine learning model documentation, investigating responsible and accountable practices. Their focus was on the external documentation of reusable ML models and services. They also proposed a documentation tool to enable data scientists to create and update user-oriented documentation effectively.

While existing research on automatic software and API documentation generation and augmentation has mainly focused on software engineering, these studies are often conducted without considering the underlying causes of documentation-related issues. Despite the increasing use of software code in machine learning, there remains a significant gap in the literature regarding the automatic generation or augmentation of machine learning documentation. Although some studies have begun to explore documentation for datasets and models in machine learning, no specific research has been conducted on machine learning API documentation. This gap highlights the critical need for a study that not only addresses the automated augmentation of machine learning API documentation but also tackles the underlying causes of documentation-related issues.

3.4 Empirical Data

In this section, we discuss the data collection processes and datasets used to fine-tune two binary classifiers and answer the research questions.

3.4.1 Documentation Related Questions

The first classifier categorizes questions as "documentation-related" or "not documentation-related" (see Section 3.5.1). To create a dataset to finetune a binary classifier, we first extracted TensorFlow API documentation-related questions from Stack Overflow using *documentation, document, doc, official, website* keywords (see Section 3.2) along with the TensorFlow and Python tags, via Stack Exchange Data Explorer.

As a result, we retrieved 24,739 distinct Stack Overflow questions from the period of January 2016 to July 2023. We then preprocessed the question bodies to remove instances where the specified keywords appeared within code snippets, possibly as variables or comments. This involved extracting text only within `<p>` tags using the `BeautifulSoup` library. Subsequently, another round of keyword matching yielded 9021 questions.

However, it's important to note that not all questions containing the specified keywords are necessarily related to TensorFlow API documentation. To address this, we randomly sampled 2,000 questions, which two annotators manually inspected and categorized as either "documentation-related" or "not-documentation-related". Both annotators have over a year of professional software engineering experience and hold M.Sc. degrees. We also removed noisy questions containing error messages and those unrelated to TensorFlow Python API documentation based on keywords: *TensorFlow JS, TF JS, TensorFlow Lite, TFX, TensorFlow.js, tfjs, C++, Java, JavaScript, C#, Swift, TensorFlow tutorial, and TensorFlow guide*. The annotation process achieved a Kappa agreement of 0.70 between the two annotators and then the annotators discussed and resolved any disagreements. This resulted in 507 TensorFlow API documentation-related questions. We then conducted various text preprocessing procedures on these 507 questions. First, we extracted all text within `<p>` tags, iteratively analyzing and removing content within `<table>`, `<tr>`, `<td>`, and `<section>` tags. Using regular expressions, we further removed any residual HTML artifacts. Finally, we combined the words with appropriate spacing, ensuring the absence of superfluous spaces or unnecessary characters in the finalized text.

3.4.2 Code Examples Required Questions

The second classifier determines whether a question requires a code example in the response, classifying questions as "example required" or "example not required" (see Section 3.5.2). To prepare the dataset for fine-tuning this binary classifier, we used the dataset created in Section 3.4.1, which includes 507 TensorFlow documentation-related questions identified through manual analysis.

Two annotators then manually reviewed each question to determine the question requires a code example in the response, assigning it one of two labels: "example required" or "example not required". They used closed card sorting [195] to categorize the questions into these two predefined categories. One annotator has over a year of professional experience as a software engineer and holds a master's degree, while the

other holds a bachelor's degree in computer science. The annotation process achieved a Kappa agreement of 0.75 between the two annotators, indicating substantial agreement [102]. The annotators then discussed and resolved any disagreements, resulting in a final dataset of 215 TensorFlow API documentation-related questions requiring code examples and 292 not requiring them.

3.4.3 TensorFlow API documentation-related questions with accepted answers

To create a dataset of TensorFlow documentation-related questions on Stack Overflow with accepted answers, we used the dataset from Section 3.4.1, which contains 507 unique TensorFlow API documentation-related questions. We used the Stack Exchange API to extract answers and their metadata based on the question IDs. For each question ID, we retrieved the answer body, creation date, answer ID, post link, and acceptance status. We determined whether the answers were accepted by checking the `is_accepted` parameter in the API response. As a result, we identified 32 TensorFlow documentation-related questions with accepted answers. According to Yang et al. [191], accepted answers on Stack Overflow are regarded as reliable solutions that users prefer for addressing problems. Therefore, we considered the accepted answers for the corresponding questions as the ground truth in this study.

3.5 Methodology

In this section, we explain each methodological steps and the functionalities.

TensorFlow uses the `Docstrings` in its source code[1] to generate API documentation[2]. A `Docstring` is a literal string written as the initial statement in a module, function, class, or method definition, and it supports markdown [157]. It describes the functionality and purpose, lists the parameters or arguments, outlines the return values, details

[1] https://github.com/tensorflow/tensorflow
[2] https://www.tensorflow.org/api_docs/

any exceptions, and offers relevant examples [131]. A generator developed by TensorFlow is used to create the HTML version of the official API documentation using the `Docstrings` in the source code file [157]. Moreover, TensorFlow allows developers to contribute to its API documentation by providing examples. This process involves forking the TensorFlow repository, editing the `Docstring` in the source file, and submitting the modifications through a pull request. The submitted pull request is then reviewed by TensorFlow maintainers. Upon approval, the changes are merged into the official documentation.

DocChameleon sits between the TensorFlow source code and official API documentation. Its objective is to enhance the API documentation by addressing documentation-related questions on Stack Overflow after a major TensorFlow release. This includes providing explanations and code examples while recommending related YouTube tutorials and Stack Overflow Q&As to help users acquire API knowledge. API knowledge refers to the domain concepts associated with the API, including terminology, usage patterns, and facts about the API's execution and runtime behavior [160]. To ensure DocChameleon operates solely on documentation-related questions, we used a binary classifier to identify such questions. Figure 3.2 illustrates the integration of DocChameleon and the binary classifier within the framework of the TensorFlow API documentation mechanism.

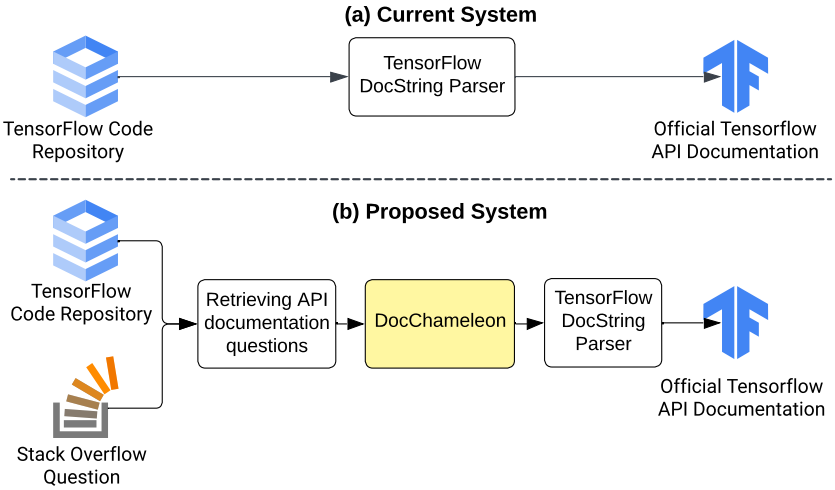


FIGURE 3.2: (a) Current system: Existing TensorFlow API documentation mechanism. (b) Proposed system: DocChameleon's position within TensorFlow API documentation mechanism.

3.5.1 Identifying documentation related questions

DocChameleon is designed to augment the API documentation based on the Stack Overflow questions that are directly relevant to the API documentation. To identify these questions, we fine-tuned a pretrained language model (PLM) as a binary classifier. We used SetFit (Sentence Transformer Fine-Tuning) [167], an efficient, prompt-free framework for fine-tuning sentence transformers with minimal labeled data through few-shot learning. This approach uses contrastive training to generate positive and negative pairs from in-class (same class) and out-class (different class) instances[3]. Fine-tuning PLMs that are pretrained on large, unannotated datasets has proven effective for downstream tasks [175]. To improve performance with limited labeled data, we fine-tuned the `stackoverflow-mpnet-base`[4] PLM pretrained on 18,562,443 title and body pairs from Stack Overflow software engineering data.

During fine-tuning, we employed Hugging Face's "unique" sampling strategy within the SetFit framework, which generates distinct positive and negative pairs by selecting all possible pairings exactly once. However, this strategy does not guarantee an equal number of positive and negative pairs when the dataset is imbalanced. Therefore, we used the dataset created in Section 3.4.1, containing 507 TensorFlow documentation-related questions, and added 493 non-documentation-related questions identified during manual analysis, resulting in a dataset of 1,000 records. To balance the dataset, we randomly sampled 400 records from each category, creating a training dataset of 800 records, while the remaining 200 records were used as a test set. With this balanced dataset, the "unique" sampling strategy generated 205,120 distinct positive and negative pairs.

We used an Nvidia P100 16GB GPU to fine-tune the sentence transformer. Following Hugging Face's SetFit documentation, we used the default learning rate of $2e-5$ and fine-tuned for one epoch. We observed that since the `stackoverflow-mpnet-base` model was pretrained on a large amount of Stack Overflow data, fine-tuning for more than one epoch led to overfitting. For fine-tuning the classification head, we again used SetFit's

[3] https://huggingface.co/blog/setfit

[4] https://huggingface.co/flax-sentence-embeddings/stackoverflow_mpnet-base

TABLE 3.1: Test results of the binary classifier for identifying documentation-related questions

Binary Classifier (Test Results)			
Accuracy	Precision	Recall	F1-Score
80.87%	82.48%	81.87%	82.17%

default learning rate and number of epochs, which are 1e-2 and 16, respectively. Despite the limited labeled dataset, the SETFIT method achieved high accuracy, precision, recall, and F1-score on the test set, as shown in Table 3.1.

3.5.2 DOCCHAMELEON

DOCCHAMELEON consists of five main steps as shown in Figure 3.3, with each step annotated by (x).

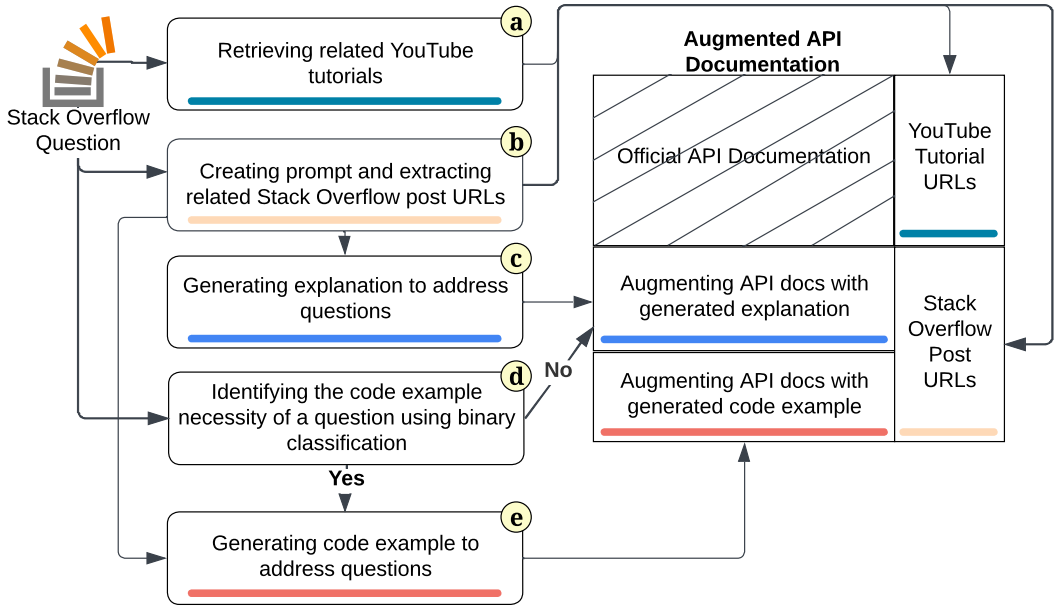


FIGURE 3.3: High-level Systematic & Functional Architecture of DOCCHAMELEON

Retrieving related YouTube tutorials: In Step (a), we augment the API documentation with related YouTube tutorials that provide both fundamental API knowledge and API-related "how-to-do" information, which can be used to gain the necessary background knowledge to address a question.

Creating prompt using Stack Overflow (SO) questions and extracting SO post URLs: In Step (b), we create a prompt by retrieving relevant API knowledge from Stack

Overflow based on the question, to be used as the context in the prompt. We then populate the context and question variables in the prompt with the retrieved context, question title and question body. Additionally, we augment the API documentation with the Stack Overflow post URLs that are used to extract relevant API knowledge for the context.

Generating explanations to address questions: In Step ⓒ, we generate a detailed explanation addressing the question based on the context retrieved from Stack Overflow, ensuring the explanation is supported by the context.

Identifying code example necessity: In Step ⓓ, we identify whether the question requires a code example by performing binary classification using a fine-tuned pretrained language model (PLM).

Generating code examples to address questions: In Step ⓔ, we generate executable code examples to address the question based on the context retrieved from Stack Overflow, regenerating iteratively if execution fails. This step is performed if the binary classifier in Step ⓓ determines that a code example is required.

Retrieving related YouTube tutorials

When developers share their knowledge with other developers through a text-based approach (documentation), it often misses background knowledge needed to fully understand workflows and processes [103]. Video tutorials that offer introductions and practical demonstrations of various tools and technologies have become popular among developers, as they enhance learning outcomes through multimedia [129, 106]. As the largest online video database, YouTube provides access to a wide range of user-created tutorials on diverse topics [68]. Therefore, we leveraged YouTube to gather related tutorial videos, where "related" refers to tutorials that can be used to gain knowledge to address a question.

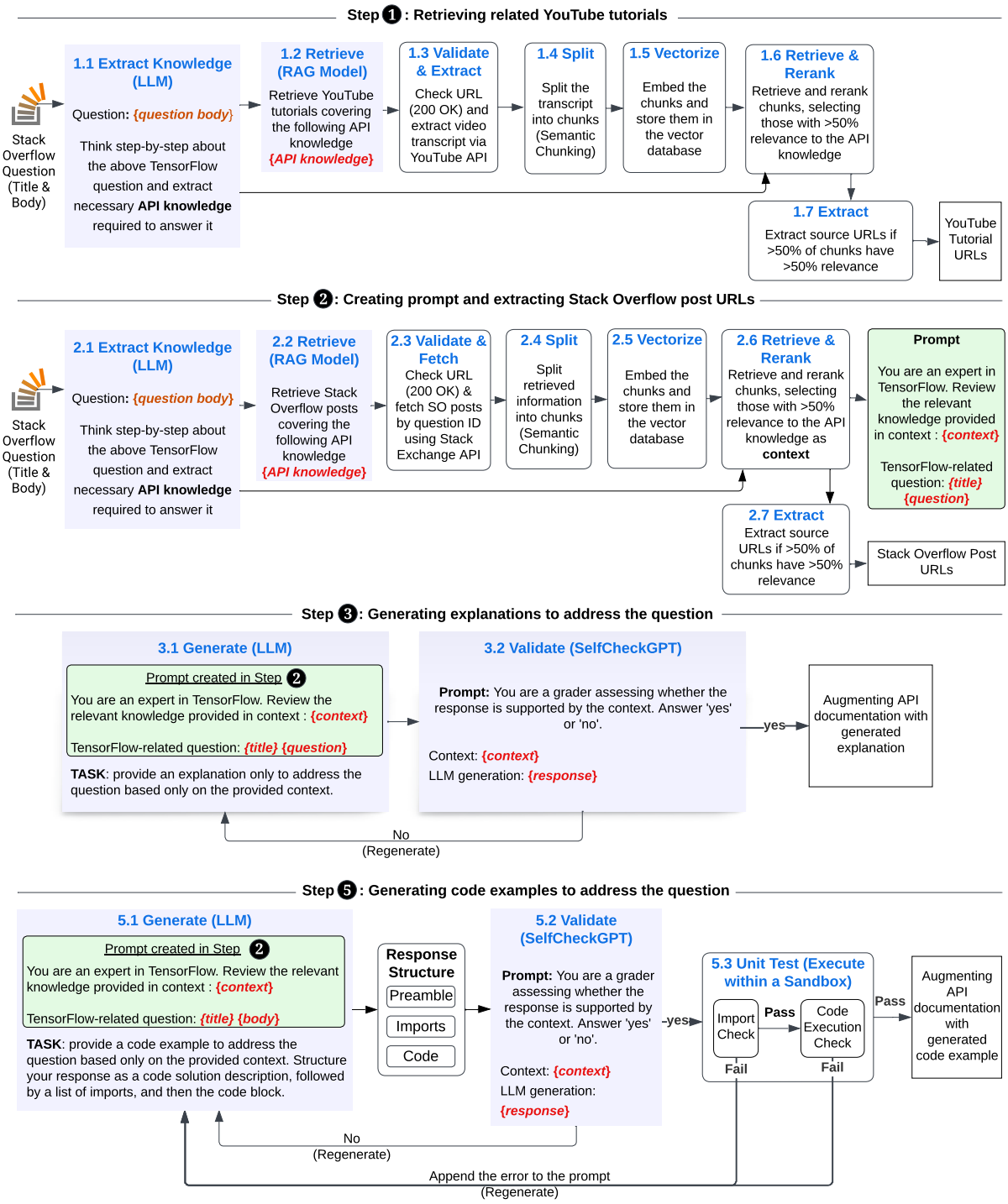


FIGURE 3.4: Internal functionality diagram of each step of DOCCHAMELEON

In Step ❶, to retrieve related YouTube tutorials, we first extract the required API knowledge needed to answer a question using the `GPT-4o` model and a zero-shot chain-of-thought (CoT) prompt in Step 1.1. This approach is supported by a study on prompt

engineering techniques conducted by Kojima et al. [82], which has demonstrated that zero-shot chain-of-thought prompting improves the reasoning capabilities of LLMs.

To retrieve relevant YouTube tutorials based on the knowledge extracted in Step 1.1, we used the `Command R+` model, Cohere's instruction-following LLM specifically designed for complex RAG pipelines. This model outperforms similar models in RAG tasks and tool usage. We employed `Command R+` because it includes a built-in `web-search` connector, enabling internet searches to retrieve relevant information along with source citations (URLs) [55]. `LangChain` is an open-source framework for developing applications using LLMs. It allows combining a prompt, data sources, and an LLM in a chain to perform a sequence of operations, and it is widely recognized in the AI community for its seamless integration with various data sources and applications [163]. Additionally, `LangChain` provides the `CohereRagRetriever`[5] object, which connects the `Command R+` model to data sources and allows for restricting its search space for information retrieval. We restricted the search space by using the `CohereRagRetriever`'s "site" option and setting it to `https://www.youtube.com/`. Upon feeding the query shown in Step 1.2 as the input to the RAG model, it search and retrieves related YouTube video URLs. We then verify the validity of each URL using the Python `request` library, ensuring a successful `HTTP 200 OK` response.

To ensure the relevance of the video tutorial to the question, we first obtained the transcript of each video using the YouTube transcript API. Then employed the semantic chunking technique to split the transcript into chunks. It uses embedding similarity to dynamically identify breaks between sentences. This ensures that each chunk contains semantically related sentences [1]. Thus, we employed `LangChain`'s `semantic chunker` to divide the transcript of each video into semantically similar chunks by evaluating embedding distances between sequential sentences. An embedding distance exceeding the 95^{th} percentile threshold (`LangChain`'s default value) indicates the beginning of a new semantic section.

We then embed the chunks using the Cohere `embed-english-v3.0` model and store

[5] https://python.langchain.com/v0.2/docs/integrations/retrievers/cohere/

them in the Chroma in-memory vector database for retrieval. Next, we use the Cohere `rerank-english-v3.0` model to re-rank the retrieved chunks based on their relevance to the API knowledge extracted in Step 1.1. Finally, we returned the URLs of the YouTube tutorials only if more than 50% of the chunks in each video transcript had a relevance score above 50%.

Creating prompt and extracting Stack Overflow post URLs

Augmenting the prompt with relevant context enables the LLM to generate more factually accurate answers during inference [40]. In Step ❷, we designed a prompt with three variables to incorporate relevant context, the Stack Overflow question title, and the question body as inputs. We applied the role-based prompting technique, which assigns the LLM a specific expert role when handling domain-specific data, thereby clearly defining domain boundaries. This can be achieved by introducing the role at the beginning of the prompt [177]. The prompt includes the `{context}` variable, which is used to add relevant contextual information through retrieval, facilitating the generation of factually accurate responses. Additionally, the prompt contains the `{title}` and `{question}` variables, which are populated with the title and body of the TensorFlow documentation-related Stack Overflow question. We selected Stack Overflow as the source for retrieving relevant context because it offers comprehensive API knowledge, including domain concepts, usage patterns, arguments, execution, and runtime behavior [145]. Additionally, it hosts a large number of highly-rated TensorFlow-related posts [70], which can be used to populate the `context` variable in the prompt by extracting relevant information to address a given question.

To retrieve relevant API knowledge as context from Stack Overflow in Step ❷, we first extract the API knowledge needed to answer a question based on the Stack Overflow question body using the `GPT-4o` model with the zero-shot chain-of-thought (CoT) prompt shown in Step 2.1. We then used the same `CohereRagRetriever` and `command-r-plus` model from Step ❶ (see Section 3.5.2) to retrieve relevant API knowledge from Stack Overflow by setting the "site" option provided in `CohereRagRetriever`

to `https://www.stackoverflow.com/`. Upon inputting the query shown in Step 2.2, the RAG model exclusively searches Stack Overflow and retrieves URLs of relevant posts. We then verify the validity of each URL using the Python `requests` library, ensuring a successful `HTTP 200 OK` response. Next, we utilized the `urlparse` Python library to parse the URLs into their components: scheme (e.g., `https`), netloc (e.g., `www.stackoverflow.com`), and path (e.g., `/questions/78781272/...`). We then extract the question ID from the path and, leveraging the Stack Exchange API, we retrieve the entire post, including the question, answers, and comments. Subsequently, we select posts with accepted answers by ensuring the `is_accepted` attribute is true. To ensure the prompt is augmented with only highly relevant information, we leveraged the same techniques used in Step ❶. We divide the text into semantically similar chunks using `semantic chunker`. We then embed the chunks using the Cohere `embed-english-v3.0` model and store them in the Chroma in-memory vector database for retrieval. Next, we use the Cohere `rerank-english-v3.0` model to re-rank the retrieved chunks based on their relevance to the API knowledge extracted in Step 2.1. To further ensure the `{context}` variable in the prompt shown in Step ❷ is populated with relevant API knowledge, we include only the chunks with a relevance score above 50%. Lastly, we populated the `{title}` and `{question}` variables in the prompt using the title and body of the corresponding Stack Overflow question, respectively.

To extract the URLs of relevant Stack Overflow posts, we return the URL of a post if more than 50% of its chunks have a relevance score above 50%.

Generating explanations to address questions

Based on the taxonomy of TensorFlow documentation-related issues, which we developed by analyzing the underlying causes of TensorFlow documentation-related questions on Stack Overflow (see Section 3.2), we observed that 28.00% of these questions are related to documentation ambiguities, where users request explanations to resolve these ambiguities. Therefore, in Step ❸, we generate an explanation to address the TensorFlow API documentation-related question by prompting the `GPT-4o` model.

We first refine the prompt from Step ❷ into a task-specific prompt by adding clear instructions on what to do and how to do it. Specifically, we direct the LLM to generate an explanation that addresses the TensorFlow documentation-related question using only the relevant API knowledge provided as context. This is done by including the task, "*provide an explanation only to address the question based on the provided context*", as shown in Step 3.1. To ensure consistency and minimize randomness, we set the `temperature` parameter to zero [136]. Finally, we feed the prompt to the `GPT-4o` model to generate the explanation.

Despite using RAG, there remains a risk of generating out of context responses, known as hallucinations [54]. To mitigate this, we employed SELFCHECKGPT [104], a zero-resource hallucination detection technique. As suggested by Manakul et al. [104], hallucinations can be effectively detected by prompting the LLM multiple times. We used a similar prompt to the one introduced in the original study, as shown in Step 3.2. The LLM is prompted four times because, as demonstrated by Manakul et al. [104], the performance of SELFCHECKGPT improves steadily as the number of times the LLM is prompted with the same input increases. However, their study found that beyond four prompts $(N = 4)$, the performance gains begin to diminish. Therefore, if all four responses are "yes", the generated explanation is used to augment the API documentation; otherwise, the regeneration process is initiated.

Identifying code example necessity

To systematically identify the necessity of a code example for a question, in Step ❹ we employ a fine-tuned PLM as a binary classifier. Using a similar method described in Section 3.5.1, we employed the SETFIT (Sentence Transformer Fine-Tuning) framework [167], a prompt-free approach for efficiently fine-tuning sentence transformers with minimal labeled data through few-shot learning. This method leverages contrastive training to create positive and negative pairs from instances within the same class (in-class) and different classes (out-class)[6]. The process consists of two steps: first,

[6] https://huggingface.co/blog/setfit

the sentence transformer is fine-tuned using contrastive learning on the input data, generating positive and negative pairs. Then, the classification head is fine-tuned using the encoded data produced by the fine-tuned sentence transformer from the first step [167].

Wang et al. [175] demonstrated that fine-tuning pre-trained language models (PLMs) on large amounts of unannotated data through self-supervised learning leads to increased performance in downstream tasks. We used the `stackoverflow-mpnet-base`[7] model, which is pre-trained on Stack Overflow software engineering data. This model was trained on 18,562,443 title and body pairs from Stack Overflow.

The fine-tuning process used a "unique" sampling strategy provided by Hugging Face that generates distinct positive and negative data pairs when using the SetFit framework. This strategy selects all possible pairings exactly once, avoiding oversampling or undersampling. However, it does not ensure an equal number of positive and negative pairs if the dataset is unbalanced. To address this, we used the dataset from Section 3.4.2, which contains 215 TensorFlow documentation-related questions that require code examples and 292 that do not. We then created a balanced dataset by randomly sampling 150 records from each category, resulting in a balanced training dataset of 300 records and a testing set of 207 records. With this balanced dataset, the "unique" sampling strategy generated 44,850 distinct positive and negative training pairs.

We fine-tuned the sentence transformer using an Nvidia P100 16GB GPU. Following the Hugging Face documentation on SetFit, we used the default learning rate of $2e-5$ and fine-tuned for one epoch. Since the `stackoverflow-mpnet-base` model was pre-trained on a large amount of Stack Overflow data, we found that increasing the epoch count beyond one led to overfitting the model. For fine-tuning the classification head, we used Hugging Face's default learning rate of $1e-2$ and fine-tuned for 32 epochs. Despite the limited labeled dataset, using the SetFit framework with a pre-trained model on Stack Overflow data enabled us to achieve high accuracy, precision, recall, and F1-score on the unseen test set of 207 records. Testing results for these metrics are shown in Table 3.2.

[7] https://huggingface.co/flax-sentence-embeddings/stackoverflow_mpnet-base

TABLE 3.2: Test results of the binary classifier for identifying code example requirement in questions

Binary Classifier (Test Results)			
Accuracy	Precision	Recall	F1-Score
88.89%	85.69%	85.67%	85.68%

Generating code examples to address questions

Our analysis of TensorFlow documentation-related questions on Stack Overflow (see Section 3.2) revealed that 64.30% of these issues are related to inadequate examples. Therefore, in Step ❺, we prompt the `GPT-4o` model to generate executable code examples that directly address the TensorFlow API documentation-related questions employing the self-corrective coding assistant developed by LangChain[8]. It is an iterative process for code generation with two unit tests (check imports and code execution) and the generated code example contains three sections: *preamble (code explanation), imports, and code body*. This code example generation process is activated only if the binary classifier in Step ❹ classifies the question as a code example required question. To facilitate safe execution of the generated code, we implemented an isolated container on Modal[9] with a Python environment, including the most frequently used machine learning and deep learning libraries (`TensorFlow`, `Keras`, `NumPy`, `Pandas`, and `Scikit-learn`) [162]. We used a container within a sandboxed Python environment because generated code can be incorrect and pose security risks. The isolated environment prevents potentially harmful or unintended code from impacting the host system or data, containing crashes, infinite loops, and other issues within the sandbox [38]. We selected the Modal platform because it uses the gVisor container runtime, which enhances security by creating a security boundary between the host and its containers through resource emulation. Additionally, Modal provides access to NVIDIA GPUs, which accelerate TensorFlow code execution [112].

[8] https://langchain-ai.github.io/langgraph/tutorials/code_assistant/langgraph_code_assistant/

[9] https://modal.com/

When the binary classifier in Step ❹ determines that the question requires an example, we refine the prompt from Step ❷ into a task-specific prompt by adding clear instructions on what to generate and how to structure the output. Specifically, we instruct the LLM to produce a code example that addresses the TensorFlow documentation-related question using only the relevant API knowledge provided as context. We also specify how to format the response by including the following task: "*Provide a code example to address the question based on the provided context. Structure your response as a code solution description, followed by a list of imports, and then the code block*", as shown in Step 5.1. To ensure consistency and reduce randomness, we set the `temperature` parameter to zero [136]. Finally, we use the `GPT-4o` model to generate the code example. To verify whether the generated response is supported by the context, we used the same SELFCHECKGPT technique employed in Step ❸. If all four responses generated by prompting the LLM using the prompt shown in Step 5.2 are "yes", the generated code is then tested for executability. If any response is "no", the LLM is prompted again for regeneration.

Subsequently, the self-corrective coding assistant uses the full generated code example, including import statements and the main code body. It first compiles and executes the import statements within the isolated Python environment we created. If this execution fails, the coding assistant updates the initial prompt in Step 5.1 by appending the error message and regenerating the code example. Upon successful execution of the *import* block, the *imports* and the *code body* are combined, and the coding assistant executes the entire code example. If this execution fails, the error message is again appended to the prompt from Step 5.1, and the code is regenerated. This iterative process of generation and execution testing continues for up to three iterations if necessary. We limited the regeneration process to three iterations in this study due to the cost constraints of using OpenAI's commercial `GPT-4o` model. If the code still fails to execute after three iterations, that code example is excluded. Only code examples that successfully execute within the three iterations are used to augment the TensorFlow API documentation.

3.6 Evaluation

We conducted both quantitative and qualitative experiments to address the following research questions (RQs). Figure 3.5 illustrates the evaluation plan for the content added to augment the official TensorFlow API documentation:

RQ1: To what extent DocChameleon can retrieve relevant information from Stack Overflow, compared to state-of-the-art API augmentation methods?

We compute and compare the relevancy of the information retrieved by DocChameleon *from Stack Overflow and those from existing methods.*

RQ2: To what extent DocChameleon can generate correct, similar and faithful explanations to augment API documentation?

To evaluate this, we calculated the answer correctness, semantic similarity, and faithfulness of the explanations generated by DocChameleon *in response to questions, using the accepted answer explanations from the corresponding Stack Overflow questions as the ground truth.*

RQ3: To what extent DocChameleon can generate executable, correct, similar and faithful code examples to augment API documentation?

We initially evaluated the executability of the generated code examples by executing those in a Python environment and then we calculated the answer correctness, semantic similarity, and faithfulness of the code examples generated by DocChameleon in response to questions, using the code examples in accepted answers from the corresponding Stack Overflow questions as the ground truth.

RQ4: How do developers perceive the enhanced API documentation by DocChameleon?

We conducted a user study aimed at assessing developers' perspectives on the relevance and accuracy of the explanations and code examples, as well as the relevance and helpfulness of the YouTube tutorial URLs and Stack Overflow post URLs included by DocChameleon *in the API documentation.*

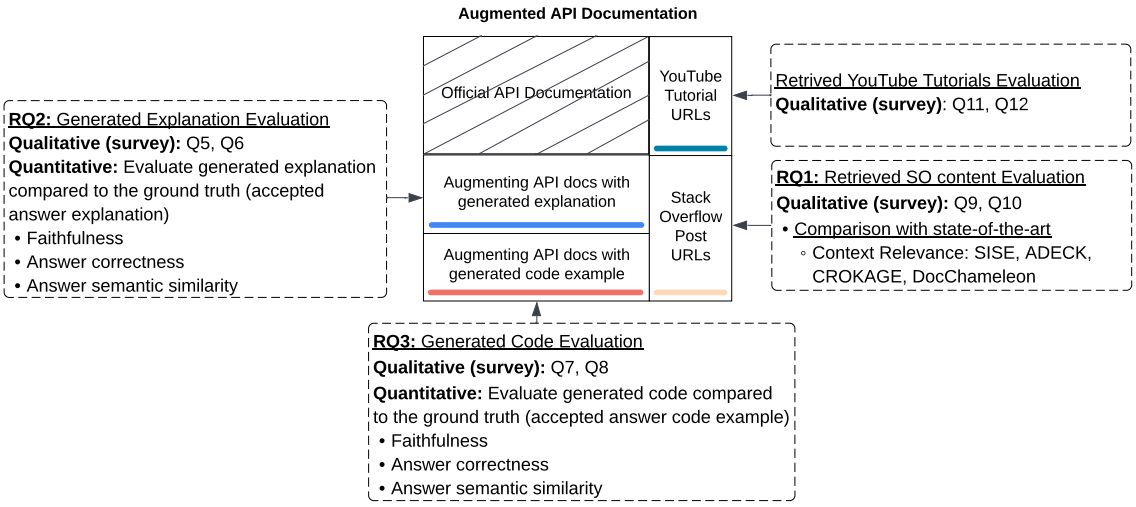


FIGURE 3.5: Evaluation plan for the content added to augment the API documentation.

3.6.1 Relevancy of the Stack Overflow information retrieved by DOCCHAMELEON (RQ1)

In the existing literature, the retrieval of relevant information from Stack Overflow for API augmentation has primarily focused on Java APIs [166, 151, 190]. To analyze the relevance of information retrieved by DOCCHAMELEON, we compared it against three baseline methods that also retrieve relevant information from Stack Overflow. Since these previous studies focused on Java APIs, we prepared our dataset using the benchmark dataset created by Silva et al. [151], which contains 115 Java API-related questions. We selected questions that explicitly mentioned the Java API name and further narrowed the selection to those related to the 15 most commonly used Java APIs [166], resulting in a final dataset of 50 questions.

Comparative Baselines:

For evaluation, we assessed the inclusion of relevant information to address a question by determining the relevance of explanations generated by DOCCHAMELEON compared to three baseline approaches.

- SISE [166] extracts insightful sentences with API knowledge from Stack Overflow posts. It retrieves Stack Overflow data using Stack Exchange API and employs

a Support Vector Machine (SVM) to identify insightful sentences based on both syntactic and semantic features.

- ADECK [190] identifies high-quality code samples and corresponding usage scenarios from Stack Overflow dump. It combines questions with their answers to create Q&A pairs, from which it extracts the usage scenario from the question title and the corresponding code sample.

- CROKAGE [151] takes a programming question as input and delivers a comprehensible code example solution. It uses Stack Overflow dump to retrieve relevant code examples written by human developers, employing a fine-tuned information retrieval technique.

Metrics:

In RQ1, we evaluate to what extent the information retrieved from Stack Overflow contains the relevant details needed to address a given question. Recent studies have shown that LLMs can be used for evaluations, demonstrating higher agreement with human experts [193] and a strong correlation between LLM and human relevance judgments [171]. RAGAS[10] is a framework that provides a set of LLM-based metrics, eliminating the need for ground truth human annotations [48]. We selected the `Context Relevancy` metric provided by the RAGAS framework because it measures how well the retrieved information contains the necessary details to address a question. Equation 3.1 shows the formula used to compute `Context Relevancy`, where $|S|$ refers to the total number of sentences in the retrieved content that are relevant to answering the given question.

$$Context\ Relevancy = \frac{|S|}{Total\ number\ of\ sentences\ in\ context} \tag{3.1}$$

Equation 3.1 evaluates the contextual relevance between the question and the information retrieved from Stack Overflow by each method. It does this by estimating the value of $|S|$, which represents the number of sentences in the retrieved content that are

[10] https://docs.ragas.io/en/stable/index.html

relevant to answering the given question. This produces a value between zero and one, with higher values indicating greater relevance between the retrieved information and the question.

3.6.2 Correctness, semantic similarity, and faithfulness of the explanations generated by DocChameleon (RQ2)

To evaluate correctness, semantic similarity, and faithfulness, we used the dataset we created in Section 3.4.3, which includes 32 TensorFlow API documentation-related questions with accepted answers. We then manually analyzed the accepted answers and identified that 26 of these contained explanations. We used the explanations in the accepted answers as the ground truth for our analysis. We chose to use accepted answers as the ground truth because they are widely regarded as reliable solutions that users prefer when addressing problems [191].

Metrics:

In RQ2, we assess the correctness, semantic similarity, and faithfulness of the explanations generated by DocChameleon in response to questions, using their accepted answers as the ground truth. We used three metrics provided by the RAGAS framework, which considers semantics, to evaluate the generated explanations in relation to the accepted answers (ground truth) and the context retrieved in Step ❷. Additionally, using the metrics provided in the RAGAS framework eliminates the need for ground truth human annotations [48].

Answer Correctness: This metric evaluates the accuracy of the generated response by comparing it to the ground truth. It integrates both factual correctness and semantic similarity. Factual correctness evaluates how much factual information is shared between the generated answer and the ground truth, based on three concepts:

- **True Positive (TP)**: Facts or statements found in both the ground truth and the generated answer.

- **False Positive (FP)**: Facts or statements found in the generated answer but not in the ground truth.
- **False Negative (FN)**: Facts or statements found in the ground truth but not in the generated answer.

$$F1\ Score = \frac{|TP|}{|TP| + 0.5 \times (|FP| + |FN|)} \tag{3.2}$$

The final answer correctness score is calculated as a weighted average of factual correctness (as determined by the F1 Score) and semantic similarity, with the score ranging from zero to one.

Answer Semantic Similarity: This metric measures semantic similarity by calculating the cosine similarity between the generated response and the ground truth. The values range from zero to one, with higher scores indicating a greater semantic similarity to the ground truth.

Faithfulness: This metric evaluates how closely the generated answer aligns with the information provided in the context. The faithfulness score is calculated as the ratio of information in the generated answer that can be inferred from the context to the total amount of information in the generated answer. The value ranges from zero to one and higher the faithfulness lower the hallucination.

3.6.3 Executability, correctness, semantic similarity, and faithfulness of the code examples generated by DocChameleon (RQ3)

Initially, we used the dataset created in Step 3.4.3, comprising 32 TensorFlow API documentation-related questions with accepted answers, to determine whether these answers contain code examples. To achieve this, we manually examined each accepted answer and identified 20 questions with code examples in their accepted answers. These code examples were then considered as the ground truth for evaluating the code examples generated by DocChameleon.

To evaluate the executability of code generated by DOCCHAMELEON in RQ3, we used the isolated Python environment created on the Modal platform (see Section 3.5.2) to execute the generated code examples in response to 32 TensorFlow API documentation-related questions that required a code example in the response. For an execution to be considered successful, the code examples needed to compile and run without exceptions within three iterations. In Section 3.5.2, we use the self-corrective coding assistant to perform code generation up to three iterations to regenerate executable code examples in the event of execution failures. Thus, we consider a code example successfully executed if it becomes executable within three iterations. If, after three iterations, the code example still throws exceptions, it is considered an unsuccessful execution. We set a limit of three iterations to manage generation costs associated with commercial LLMs. To assess the correctness, semantic similarity, and faithfulness of the code examples generated by DOCCHAMELEON in response to questions, we used 20 TensorFlow API documentation-related questions that we identified through manual analysis as containing code examples in their accepted answers. The evaluation was conducted using the three metrics outlined in Section 3.6.2.

3.6.4 Developers' perspective on the augmented TensorFlow API documentation (RQ4)

We designed a user study to evaluate developers' perspectives on the relevance and accuracy of the generated explanations and code examples, as well as the relevance and helpfulness of the YouTube tutorials and Stack Overflow posts included by DOCCHAMELEON in the TensorFlow API documentation. First, we created a demographics questionnaire consisting of seven multiple-choice questions and two open-ended questions focusing on participants' professions, experience in software development and machine learning, and their familiarity with Python and TensorFlow. We also evaluated how often participants use official API documentation, forums (Stack Overflow and GitHub), video tutorials, online courses, interactive tutorials, webinars/live demos,

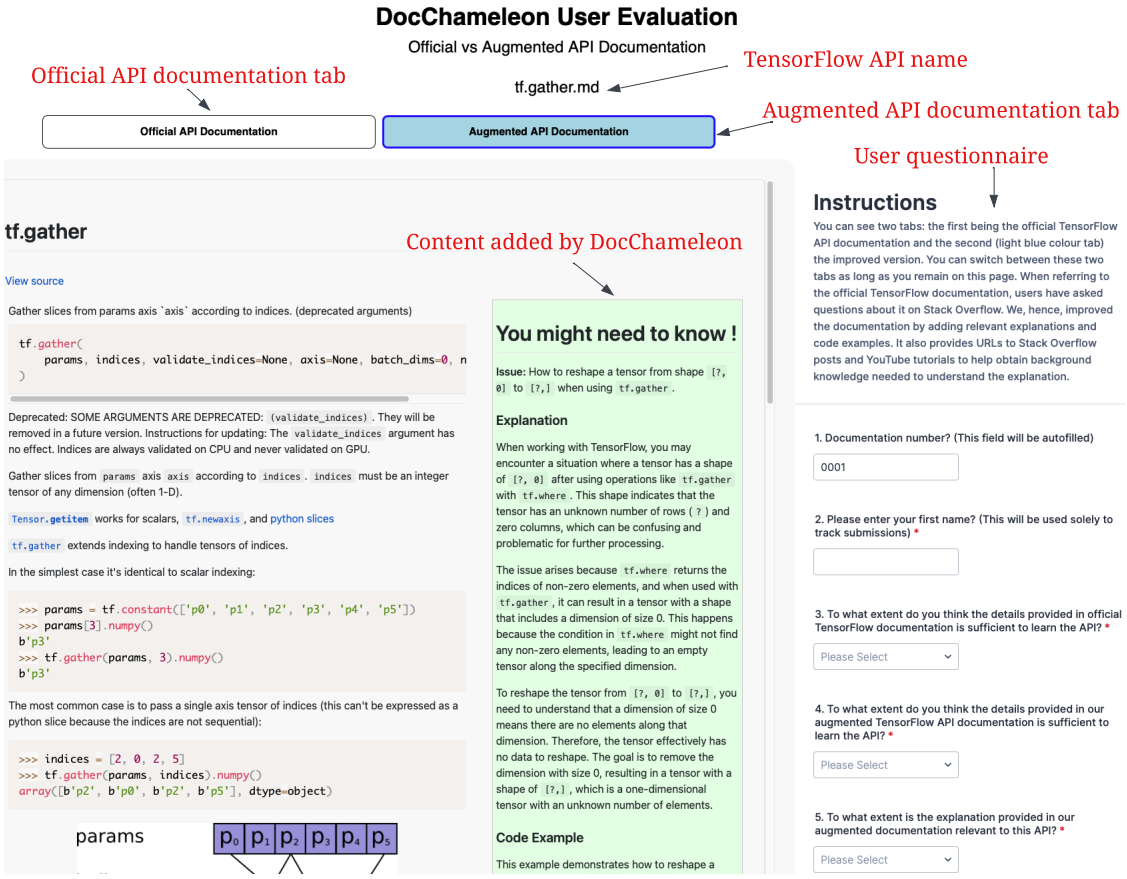


FIGURE 3.6: DOCCHAMELEON user study evaluation portal's web interface

books/e-books, and other resources during development. Finally, we analyzed how often participants refer to the TensorFlow API documentation itself, and how often they turn to YouTube tutorials and Stack Overflow posts when using the TensorFlow API documentation.

We developed an evaluation web portal for participants to compare the official and augmented TensorFlow API documentation and complete a questionnaire. Figure 3.6 shows the portal's user interface. The questionnaire included eleven multiple-choice and two open-ended questions to gather developers' perspectives on our augmented API documentation. These questions covered the sufficiency of details in both the official and our augmented TensorFlow documentation for learning the API, the relevance of explanations and code examples to the API, and their accuracy in addressing documentation issues as well as the relevance and helpfulness of YouTube tutorials and Stack Overflow posts for obtaining the background knowledge to resolve API issues.

Additionally, we inquired about their preferred documentation—whether they favor the official TensorFlow API documentation or our augmented version for programming tasks. Finally, we asked for their opinions on the most and least helpful parts of our augmented API documentation.

We recruited 18 participants, including 14 graduate students (MSc and PhD) from the computer science department who also work as research assistants, along with two software engineers, one technical systems analyst, and one engineering manager. In the dataset we created in Step 3.4.3, which contains 32 TensorFlow API documentation-related questions with accepted answers, we identified 24 unique TensorFlow APIs that these questions are about. In this user study, we analyzed both the official TensorFlow API documentation and our augmented documentation for those 24 TensorFlow APIs. Each participant was randomly assigned four augmented API documents generated by DocChameleon, along with the corresponding official documentation. Each document was reviewed by three participants, resulting in 72 submissions. To streamline the evaluation process, we assigned each augmented API document a unique number, and participants received access to the study portal via a URL. Upon accessing the URL, participants first had to complete a demographic questionnaire and were then redirected to a login page, where they entered the unique document number provided to them. Participants were asked to evaluate the DocChameleon augmented API documentation in comparison to the official TensorFlow API documentation by completing the questionnaire. After submitting their responses, participants returned to the login page to evaluate the next assigned document in the same manner.

3.7 Results and Discussions

In this section, we sequentially present the results of the four stated research questions (RQs).

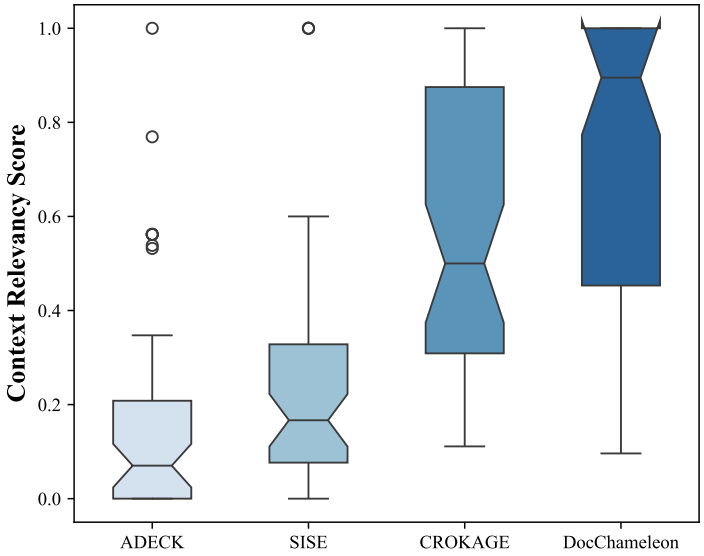


FIGURE 3.7: Comparison of the context relevancy scores of information retrieved from Stack Overflow by DOCCHAMELEON and baseline methods.

3.7.1 Relevancy of the information retrieved from Stack Overflow by DOCCHAMELEON (RQ1)

Following the evaluation metric introduced in Section 3.6.1, we assessed the retrieved information from Stack Overflow by DOCCHAMELEON with respect to the baselines in terms of `Context Relevancy`, determining the inclusion of relevant information to address a question. Figure 3.7 illustrates the notched box plot distribution of context relevance scores, showing that the `Context Relevancy` score of information retrieved from Stack Overflow by DOCCHAMELEON is significant.

We conducted a statistical comparison of the group means of `Context Relevancy` scores among the ADECK [190], SISE [166], CROKAGE [151], and DOCCHAMELEON tools using the one-way Analysis of Variance (ANOVA) test. The one-way ANOVA is a statistical method that analyzes a single independent variable to determine whether there are statistically significant differences between the means of three or more independent groups [64]. We observed a p-value of $4.78e-23$, which is less than the threshold of 0.05, indicating a significant difference between the `Context Relevancy` scores for ADECK [190], SISE [166], CROKAGE [151], and DOCCHAMELEON. However, while the

TABLE 3.3: Mann Whitney test p-values and Cohen's effect sizes for comparing `Context Relevancy` scores of information retrieved from Stack Overflow by DOCCHAMELEON against baselines.

Tools	p-Value	Effect Size
ADECK [190] vs DOCCHAMELEON	5.47e-13	+ 2.05
SISE [166] vs DOCCHAMELEON	5.43e-11	+ 1.81
CROKAGE [151] vs DOCCHAMELEON	0.01	+ 0.54

ANOVA test reveals that a difference exists, it does not specify which groups differ significantly. Therefore, to further identify the specific groups with statistically significant differences, we employed the Mann-Whitney test [148]. It is a non-parametric method employed to compare group means between two independent groups. However, the p-value alone does not indicate the magnitude of the difference beyond its significance. Thus, we used Cohen's effect size to measure the magnitude of the difference between two group means [56]. Table 3.3 shows the p-values and effect sizes associated with retrieving relevant information from Stack Overflow among the four tools.

When comparing the inclusion of relevant information in the content retrieved from Stack Overflow by DOCCHAMELEON to address a question against the baselines, we found a statistically significant difference in `Context Relevancy`, as shown in Table 3.3. This is confirmed by the visual representation in Figure 3.7, where the notched box plots for ADECK [190], SISE [166], CROKAGE [151], and DOCCHAMELEON show no overlap, clearly indicating distinct differences in their performance. The comparison of `Context Relevancy` scores between ADECK [190] and DOCCHAMELEON yielded a p-value of $5.47e-13 (< 0.05)$, indicating a significant difference and showed the largest effect size of $+2.05$ across all baselines. And also there is a significant difference in `Context Relevancy` scores between SISE [166] and DOCCHAMELEON, with a p-value of $5.43e-11$ and a large effect size of $+1.81$. The comparison between CROKAGE [151] and DOCCHAMELEON shows a p-value of 0.01 and a medium effect size, indicating only a moderate advantage of DOCCHAMELEON over CROKAGE [151]. Overall, DOCCHAMELEON consistently outperforms the baseline tools, demonstrating that the information it retrieves includes the most relevant content to address questions compared to the baselines.

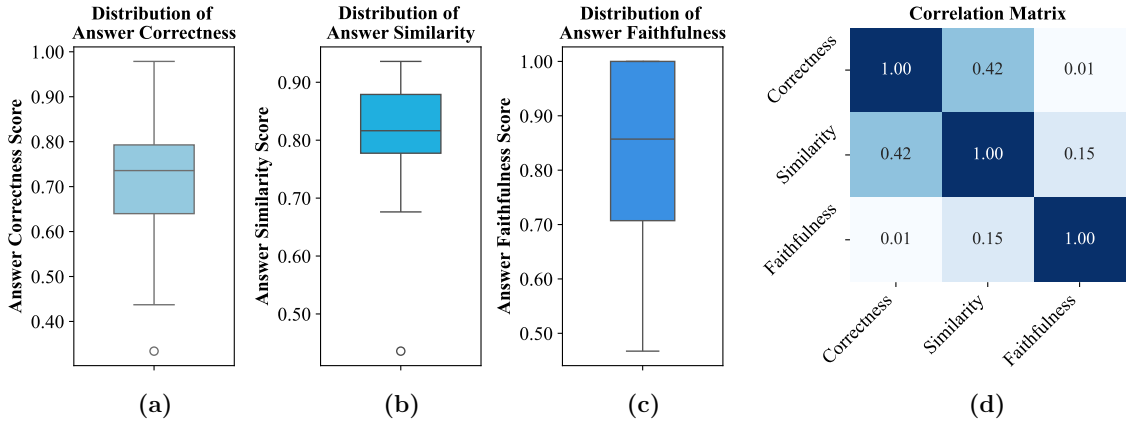


FIGURE 3.8: Distribution of (a) correctness, (b) similarity, and (c) faithfulness scores of the **generated explanations** relative to the Stack Overflow accepted answers, and (d) the correlation between correctness, similarity, and faithfulness scores.

We found that the information retrieved from Stack Overflow by DOCCHAMELEON has an overall higher `Context Relevancy` score compared to the baselines. When comparing the average `Context Relevancy` scores across the techniques, DOCCHAMELEON retrieved information was 30.35% more relevant to the questions compared to CROKAGE [151], which had the second-highest `Context Relevancy` score.

3.7.2 DOCCHAMELEON generated explanations (RQ2)

We evaluated the correctness, semantic similarity, and faithfulness of the explanations generated by DOCCHAMELEON in response to TensorFlow API documentation-related questions, comparing them to the ground truth explanations (accepted answers) using the metrics defined in Section 3.6.2. The dataset used for this analysis, as detailed in Section 3.6.2, consists of 26 TensorFlow API documentation-related questions, each with an accepted answer providing the ground truth explanation.

As shown in Figure 3.8-(a), the median answer correctness score is approximately 0.75, indicating that half of the generated explanations have a correctness score above this value. The overall distribution, represented by the whiskers extending from around 0.45 to 0.95, shows that all generated explanations achieved a correctness score above 0.45 when compared to the ground truth explanations. Based on the correctness score distribution, spanning from the first quartile (0.65) to the top whisker (0.95), shows that

75% of the generated explanations have correctness scores within this range, indicating that the generated explanations are consistently correct in relation to the accepted answers of the corresponding questions.

The results of the answer similarity scores, as shown in Figure 3.8-(b), indicate that the explanations generated by DOCCHAMELEON are generally semantically similar to the accepted answers of the corresponding Stack Overflow questions. The overall similarity score distribution, represented by the whiskers extending from approximately 0.68 to 0.95, demonstrates that nearly all generated explanations achieved a similarity score above 0.68, with a median similarity score around 0.82.

The faithfulness score measures how well the generated answers align with the information provided in the context. As shown in Figure 3.8-(c), 75% of the generated explanations achieved a faithfulness score above approximately 0.70, based on the faithfulness score distribution spanning from the first quartile (0.70) to the top whisker (1.00). The median faithfulness score of around 0.81 indicates that half of the generated explanations align very closely with the information provided in the context.

Finally, we calculated the Spearman correlation between the three metrics to investigate potential relationships. As shown in the correlation matrix in Figure 3.8-(d), there is a moderate positive correlation of 0.42 between answer correctness and similarity. This suggests that as the similarity of a generated explanation to the accepted answer increases, its likelihood of being correct also increases. In contrast, there is negligible correlation between either correctness and faithfulness (0.01) or similarity and faithfulness (0.15).

We found that the explanations generated by DocChameleon demonstrated consistent correctness, with an average score of 70.11% compared to the accepted answers. Additionally, the results showed that these explanations are generally similar to the accepted answers, with an average similarity score of 80.73%. DocChameleon also produced explanations that closely align with the API knowledge provided in the context, as indicated by an average faithfulness score of 82.52%, suggesting minimal hallucination most of the time. Our correlation analysis further revealed that as the similarity between the generated explanation and the accepted answer increases, the likelihood of correctness also improves.

3.7.3 DocChameleon generated code examples (RQ3)

To assess the executability of code examples generated by DocChameleon in RQ3, we executed each example in an isolated Python environment containing all the necessary libraries (see Section 3.5.2) for running TensorFlow-related code. We analyzed the number of executable code examples and the number of iterations DocChameleon needed to generate an executable example, with a limit of three iterations. Additionally, we compared the successfully executed and failed code examples by examining the lines of code (LOC) and the number of imports. Figure 3.9 shows the number of successfully executed code examples, the number of examples that failed after three iterations, and the variation in LOC and import statements in relation to the execution status of the generated code.

We observed that 18 out of 32 code examples generated by DocChameleon successfully executed, yielding an executability rate of 56.25%. Notably, 88.88% of these successful executed code examples were generated in the first iteration. Only one code example successfully executed after being regenerated in the second iteration, and another after regeneration in the third iteration. To determine if continued regeneration could make non-executable code examples executable, we extended the process to five iterations for the 14 examples that failed after three iterations. We found that all examples which failed after three iterations still failed after five iterations. We compared the

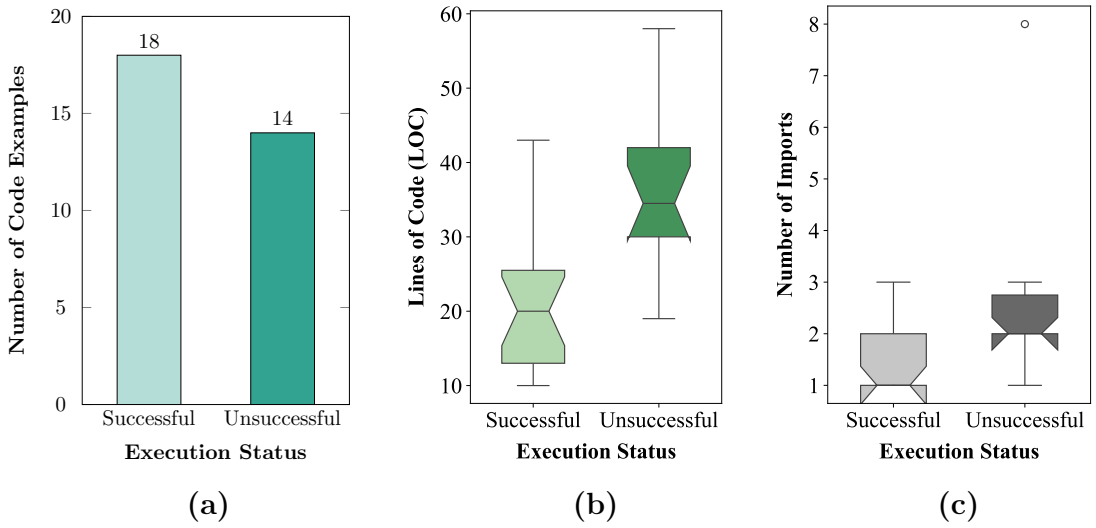


FIGURE 3.9: Distribution of (a) number of executable and non-executable generated code examples (b) lines of code (c) number of import statements in the code with respect to the execution status

variation in lines of code (LOC) between successfully executed and unsuccessful code examples to determine if there is a statistically significant difference and to measure the effect size. This analysis was conducted using the Mann-Whitney test and Cohen's effect size. The results yielded a p-value of 0.001, indicating a statistically significant difference between successful and unsuccessful executions. This finding is visually supported by Figure 3.9-(b), where the notched box plots for successful and unsuccessful executions show no overlap, clearly indicating a distinct difference in LOC with a large effect size of $+1.40$. We again computed the statistical difference and effect size between successful and unsuccessful executions, this time focusing on the number of import statements in the code examples. The analysis revealed a p-value of 0.02, indicating a statistically significant difference between the two groups, with a medium effect size of $+0.75$. This distinct difference is also visually evident in Figure 3.9-(c), where the notches of the two notched box plots show no overlap.

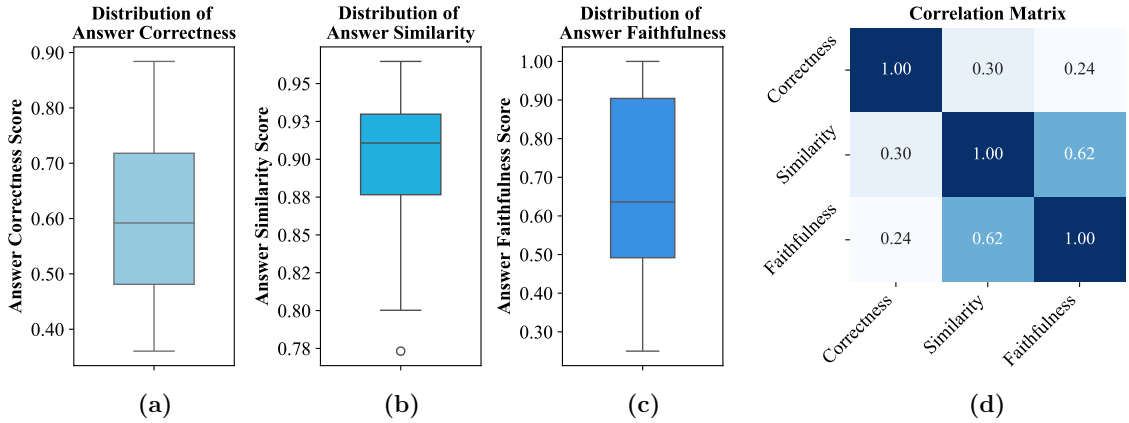


FIGURE 3.10: Distribution of (a) correctness, (b) similarity, and (c) faithfulness scores of the **generated code examples** relative to the Stack Overflow accepted answers, and (d) the correlation between correctness, similarity, and faithfulness scores.

DOCCHAMELEON generated executable code examples with a success rate of 56.25% for TensorFlow API documentation-related questions on Stack Overflow. Analysis of LOC and import statements revealed that successful executions had significantly lower LOC and fewer import statements, indicating that simpler, smaller code is more likely to run successfully.

We then computed the generated code examples correctness, semantic similarity and faithfulness with respect to the ground truth code examples using the metrics defined in Section 3.6.2 using the dataset we derived with 20 TensorFlow API documentation-related questions on Stack Overflow that require code example in the response and contain a code example in the accepted answer.

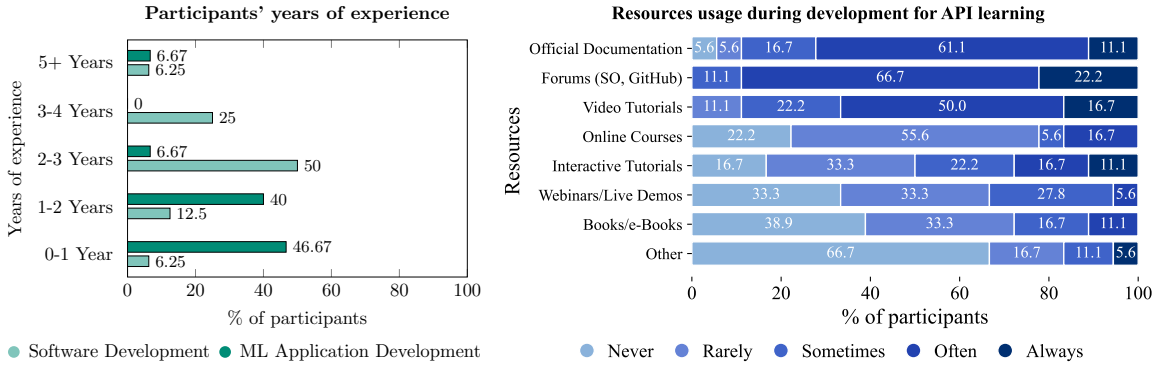
The answer correctness distribution of the generated code examples in Figure 3.10-(a) shows that half of the examples achieved a correctness score above 0.60, with a maximum score around 0.90. However, the lowest whisker point (0.35) and the first quartile (0.50) indicate that 25% of the generated code examples scored below 0.50 compared to the ground truth code examples. The distribution from the first quartile (0.50) to the top whisker (0.90) indicates that 75% of the generated code examples consistently scored a correctness score above 0.50 in comparison to the code examples in the accepted answers.

As shown in Figure 3.10-(b), all the code examples generated by DOCCHAMELEON achieved a similarity score above 0.80 and this is illustrated by the box plot, where the whiskers extend from approximately 0.80 to 1.00. These results indicate that DOCCHAMELEON consistently generates code examples that are closely similar to code examples found in the accepted answers of TensorFlow-related questions on Stack Overflow.

The distribution of faithfulness scores for the code examples generated by DOCCHAMELEON, as shown in Figure 3.10-(c), indicates that 75% of the generated code examples achieved a faithfulness score above 0.50, as evidenced by the distribution from the first quartile (0.50) to the upper whisker (1.00). However, the distribution also shows that 25% of the code examples failed to reach a faithfulness score of at least 0.50, as shown by the range from the lower whisker (0.25) to the first quartile (0.50). This indicates that a quarter of the generated code examples are poorly aligned with the information provided in the context.

We computed the Spearman correlation between the three metrics to determine whether there is any relationship between the correctness, similarity, and faithfulness scores when generating code examples. In Figure 3.10-(d), we observed a moderate positive correlation of 0.62 between similarity and faithfulness. Additionally, the correlation matrix shows a weak positive correlation of 0.30 between similarity and correctness. However, the correlation between correctness and faithfulness is negligible, with a value of 0.24.

The generated code examples demonstrated moderate correctness, with an average score of 59.67% compared to the accepted answers, and a high similarity of 89.91%, showing close similarity to the code examples in the accepted answers. However, despite an average faithfulness score of 67.43%, quarter of the examples failed to reach at least 50% faithfulness, indicating poor alignment with the information provided as context.



(a) (b)

FIGURE 3.11: Distribution of (a) participant's years of experience, (b) resource usage during development for API learning.

3.7.4 User Study Results (RQ4)

We first collected demographic information through nine questions (see Section 3.6.4). As shown in Figure 3.11-(a), half of the participants have two to three years of software development experience, while approximately 30% have more than three years. In terms of machine learning application development, 46.67% have at most one year of experience, followed by 40% with one to two years. Notably, none of the participants have three to four years of experience in this area, while 6.67% have over five years. The figure also shows that more than 75% of participants have over two years of software development experience, yet their experience in machine learning application development remains under two years.

We then asked participants about their familiarity with Python programming and the TensorFlow library. The results in Table 3.4 shows that all participants have at least some familiarity with Python, with 66.67% being very familiar and 22.23% identifying as experts, indicating that none are beginners or unfamiliar with the language. In contrast, none of the participants are experts in TensorFlow. While 11.12% are beginners and 50% are somewhat familiar, only 38.88% are very familiar with the TensorFlow library.

We then asked participants to what extent they rely on official API documentation,

TABLE 3.4: Participants' familiarity with Python programming and the TensorFlow library

	% of participants	
	Python Programming	**TensorFlow Library**
Not familiar	0	0
Beginner	0	11.12
Somewhat familiar	11.12	50.00
Very familiar	66.67	38.88
Expert	22.23	0

community forums (Stack Overflow and GitHub), video tutorials, online courses, interactive tutorials, webinars/live demos, books/e-books, and other resources during development to learn APIs. As shown in Figure 3.11-(b), more than 50% of participants often use official documentation, community forums, and video tutorials. Notably, 38.90% always rely on community forums and video tutorials, while only 11.10% consistently use official documentation. Interestingly, 5.6% never use official documentation. Additionally, most participants rarely turn to online courses, interactive tutorials, and webinars/live demos, while a significant portion (38.90%) never use books/e-books for API learning. Furthermore, while 66.70% do not use resources beyond the primary options shown in Figure 3.11-(b), around 30% use LLM applications like ChatGPT, Microsoft Copilot, and Claude for API learning.

Finally, in the demographics questionnaire, we asked participants how frequently they use TensorFlow API documentation, as well as how often they turn to YouTube and Stack Overflow posts when using the documentation. The results show that while all participants refer to TensorFlow API documentation, only 27.8% use it often, and none of the participants always rely on it. Interestingly, around 95% of participants refer to YouTube or Stack Overflow posts when using TensorFlow API documentation. Among them, the majority (55.6%) often refer to these sources, and 11.1% always refer to YouTube and Stack Overflow posts when working with TensorFlow API documentation.

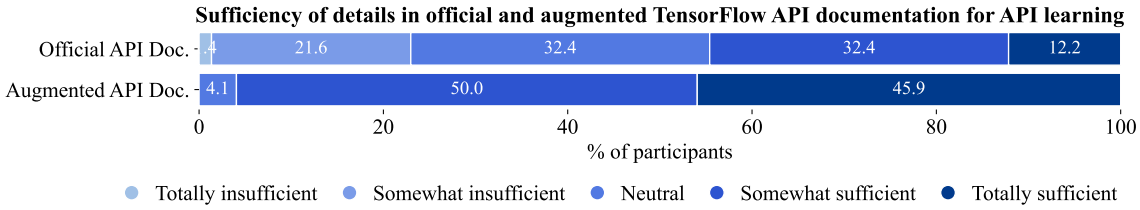


FIGURE 3.12: Distribution of participant's opinion on the sufficiency of details provided in the official and augmented TensorFlow API documentation for learning the API

We observed that over 65% of participants frequently use official documentation, community forums, and video tutorials for API learning. Notably, compared to official documentation, more participants always rely on community forums and video tutorials, indicating that the documentation alone is inadequate. Additionally, 95% of participants using TensorFlow API documentation also refer to YouTube and Stack Overflow posts, highlighting its limitations as a standalone learning resource.

In the second part of the user study, we presented participants with both the official TensorFlow API documentation and an augmented version and asked them thirteen questions (see Section 3.6.4). We first asked participants to what extent the official and augmented TensorFlow API documentation is sufficient for learning the API. As shown in Figure 3.12, half of participants found the augmented TensorFlow API documentation somewhat sufficient, while 45.90% considered it totally sufficient for API learning. In comparison, only 32.40% and 12.20% of participants rated the official TensorFlow API documentation as somewhat sufficient and totally sufficient, respectively. Additionally, while 32.40% of participants were neutral regarding the sufficiency of details in the official API documentation, 22% found it insufficient.

We then asked participants about the relevance of the explanations and code examples to the API, as well as the accuracy of these elements in our augmented API documentation in addressing API documentation-related issues. As shown in Table 3.5, 71.62% of participants stated that the generated explanations are totally relevant to the

API, while 27.02% found them somewhat relevant. Only 1.35% of participants were neutral. Similarly, 56.75% of participants found the generated explanations to be totally accurate in addressing documentation-related API issues, while 39.18% considered them somewhat accurate. For the generated code examples, 67.56% of participants reported that they are totally relevant, and 29.72% found them somewhat relevant. Regarding accuracy, around 90% of participants stated that the code examples are accurate (either totally or somewhat) in addressing documentation-related issues, while only 10.81% were neutral. Notably, none of the participants considered the generated explanations or code examples irrelevant or inaccurate.

TABLE 3.5: Developers' perspectives on the relevance of the explanations and code examples generated by DOCCHAMELEON to the API and their accuracy in addressing API issues.

		% of participants	
		Explanations	**Code Examples**
Relevancy	Totally irrelevant	0	0
	Somewhat irrelevant	0	0
	Neutral	1.35	2.70
	Somewhat relevant	27.02	29.72
	Totally relevant	71.62	67.56
Accuracy	Totally inaccurate	0	0
	Somewhat inaccurate	0	0
	Neutral	4.05	10.81
	Somewhat accurate	39.18	31.08
	Totally accurate	56.75	58.10

Next, participants were asked to evaluate the relevance and helpfulness of the Stack Overflow posts and YouTube tutorial links in our augmented API documentation for obtaining background knowledge and learning concepts related to documentation-related API issues. As shown in Table 3.6, a significant portion of participants (approximately 90%) found the Stack Overflow post links relevant (either totally or somewhat) for obtaining background knowledge to address documentation-related API issues. However, 5.40% of participants considered the Stack Overflow links irrelevant (either totally or somewhat). Additionally, 51.35% of participants rated the Stack Overflow posts as very helpful, with 33.78% considering them somewhat helpful. On the other hand, 6.75% found the posts unhelpful (either very or somewhat). For the YouTube tutorial links, 44.59% of participants found them totally relevant, while 21.62% considered them

somewhat relevant for learning concepts related to the API issues. Although none of the participants rated the YouTube tutorials as totally irrelevant, 5.40% found them somewhat irrelevant. Regarding the helpfulness of the YouTube tutorials, around 65% of participants stated that they were helpful (either very or somewhat). Only a small portion of participants (4.05%) found the YouTube tutorial links unhelpful (either very or somewhat).

TABLE 3.6: Developers' perspectives on the relevance and helpfulness of Stack Overflow posts and YouTube tutorials included by DOCCHAMELEON for obtaining background knowledge and learning concepts related to the API.

		% of participants	
		Stack Overflow Posts	**YouTube Tutorials**
Relevancy	Totally irrelevant	2.70	0
	Somewhat irrelevant	2.70	5.40
	Neutral	5.40	4.05
	Somewhat relevant	21.62	21.62
	Totally relevant	67.56	44.59
Helpfulness	Very unhelpful	2.70	1.35
	Somewhat unhelpful	4.05	2.70
	Neutral	8.10	5.40
	Somewhat helpful	33.78	21.62
	Very helpful	51.35	43.24

We further evaluated participants' preferences between the official TensorFlow API documentation and our augmented API documentation for programming tasks. A substantial majority of participants (93.24%) expressed a preference for our augmented API documentation (either slightly or totally), while 5.40% were neutral. Only 1.35% of participants indicated a slight preference for the official TensorFlow API documentation, and no participants stated that they totally prefer official API documentation.

Finally, we asked participants to identify the most and least helpful parts of our augmented API documentation. The results showed that the majority (62.5%) found code examples to be the most useful, followed by explanations (35.93%). Participants appreciated code examples for providing detailed usage information not covered in the official TensorFlow API documentation, which enhanced their understanding of the API. Additionally, around 17% of participants found YouTube tutorials most helpful, particularly those who consider themselves visual learners. However, 45% of the participants

rated YouTube tutorials as the least helpful due to their lack of specificity to API issues and the length of the videos. Furthermore, approximately 25% of participants considered Stack Overflow posts the least helpful, mentioning no direct relation to the API and low upvote counts. About 13% of participants found explanations and code examples least helpful, noting that explanations contain too many words and requested more code examples, while some code examples were seen as overly simplistic.

Compared to the official TensorFlow API documentation, 50% more participants reported that the augmented API documentation is sufficient for learning the API. Furthermore, a significant majority of participants (approximately 90%) found the explanations, code examples, Stack Overflow post links, and YouTube tutorial links added by DocCHAMELEON relevant, accurate, and helpful. Additionally, 93.24% of participants preferred the augmented API documentation over the official TensorFlow API documentation for programming tasks.

3.8 Implications and Future Works

Currently, code generation and its evaluations are mostly focused on the software engineering field, even though machine learning is increasingly used in other areas through programming. Furthermore, current methods for automated API documentation augmentation produce a single document for all users, which may not meet the varied expertise levels of different users.

3.8.1 Code generation for machine learning tasks

Code generation using LLMs has been extensively studied and LLMs have demonstrated state-of-the-art performance in the software engineering domain [92, 38, 90, 123, 53]. Additionally, research has been conducted on using RAG for code generation [179], as well as using an iterative approach to generate code, with regeneration upon failure [137]. In DocCHAMELEON, the code example generation process leverages both

the contextual information retrieved from Stack Overflow and a self-corrective coding assistant developed by `LangChain`.

In order to evaluate the functional correctness of the generated code when combining these two techniques, we employed the HUMANEVAL [38] benchmark introduced by OpenAI, which is widely used in code generation studies. It contains 164 handwritten Python programming problems, each including a function signature, `Docstring`, and several unit tests, with an average of 7.7 tests per problem. Wang et al. [179] used the HUMANEVAL dataset to demonstrate the impact of RAG on code generation by comparing five code-specific LLMs and four general LLMs. Moreover, Chen et al. [38] introduced Codex, a specialized GPT model trained for code generation, and evaluated its functional correctness using the HUMANEVAL dataset. Similarly, Luo et al. [99] introduced WizardCoder, which enhances code LLMs with complex instruction fine-tuning and assessed its performance across five closed-source and open-source models using the HUMANEVAL dataset.

To evaluate the accuracy of the generated code, we used the `pass@k` execution-based metric, which assesses functional correctness based on Equation 3.3 by executing the provided test cases in the HUMANEVAL dataset. Here, n denotes the total number of generated samples ($n \geq k$), k refers to the number of selected samples for unit testing, and c represents the number of correct samples that pass the unit tests out of the k selected samples. Due to cost constraints associated with using a commercial model, we generated only one sample ($n = k = 1$) per problem in the HUMANEVAL dataset and calculated `pass@1` accordingly. Essentially, `pass@k` measures the number of correct samples ($c \leq n$) that pass the test cases out of the total samples generated for each Python programming problem in the HUMANEVAL dataset.

$$Pass@k = \mathbb{E}_{Problems}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \tag{3.3}$$

We evaluated the executability of code generated by DOCCHAMELEON by first generating code examples based on problems from the HUMANEVAL dataset, followed by executing this code in an isolated Python environment. Upon successful execution, we

TABLE 3.7: Code generation pass@1 on HUMANEVAL

Method	Pass@1 (%)
GPT-3.5	72.60
GPT-4	75.60
GPT-3.5 + RAG	91.50
GPT-4 + RAG	92.60
GPT-4o + RAG + Self-Correction	**92.00**

proceeded to run test cases to assess the functional correctness of these code examples. Our evaluation was conducted on all 164 programming problems included in the HUMANEVAL benchmark dataset. The self-corrective coding assistant used in DOCCHAMELEON regenerates code upon failed executions, with a limit of three iterations. DOCCHAMELEON generated executable code on the first iteration for 97.5% of the problems (160 out of 164). For the remaining four problems, executable code was successfully generated in the second iteration, achieving 100% executability within a maximum of two iterations. This indicates that DOCCHAMELEON produced compilable code for all 164 problems in the HUMANEVAL dataset. We then executed the test cases provided in the HUMANEVAL dataset on the generated code, and the results show a high level of functional correctness, with a pass@1 score of 92.0%. Table 3.7 presents the pass@1 results on HUMANEVAL when evaluating different techniques and LLMs, as directly adopted from existing evaluations [179].

The results shown in Table 3.7 indicate that combining the RAG technique with a self-corrective coding assistant, which iteratively checks and regenerates code upon failure, does not improve the functional correctness of the generated code when evaluated using the HUMANEVAL dataset. However, when evaluating code generated based on machine-learning specific problems, the impact on functional correctness may vary. Despite the growing use of machine learning across various fields through programming, the literature has yet to address code generation specifically for machine learning tasks. In this study, we demonstrate that DOCCHAMELEON can generate executable code examples for such tasks with a success rate of 56.25%. However, further studies

and benchmark datasets like HUMANEVAL are necessary to evaluate the functional correctness of the generated code for machine learning tasks by testing it against relevant test cases.

3.8.2 Generative AI for dynamic API documentation augmentation

As software applications continuously expand their scope, they are being used across various fields, ranging from economics and life sciences to manufacturing and fashion design. Despite developers being familiar with the application domain, their educational backgrounds and experience in software engineering are often quite diverse. This diversity requires documentation that is accessible and understandable for users with different levels of expertise and experience. However, current API documentation techniques usually offer a single version for all users, ignoring differences in skill levels (see Section 3.3). We argue that our study slightly broadens the range of developers with different expertise and experience levels who can benefit from the API documentation. We achieve this by adding explanations and code examples that address documentation-related issues encountered by developers and by recommending relevant YouTube tutorials and Stack Overflow posts that provide the background knowledge needed to resolve API issues. However, this benefit is modest, and further research is necessary to augment API documentation in a way that aligns with users' expertise levels.

Previous software engineering literature has extensively explored methods for assessing user expertise, often leveraging platforms like Stack Overflow [178, 172, 188, 16, 102]. Our study also demonstrated the potential of leveraging generative AI to augment API documentation based on documentation-related questions. Researchers can further build on our work by integrating functionalities that identify users' expertise levels and augment the API documentation accordingly.

3.9 Threats to Validity

The metrics used in our study could introduce potential *construct validity* concerns regarding the results. We used the RAGAS framework [48], which offers LLM-based metrics for assessing context relevancy, answer correctness, answer semantic similarity, and faithfulness scores. This choice was based on recent studies demonstrating strong agreement between LLMs and human experts [193] and a high correlation between LLM and human judgments [171]. Although alternative evaluation frameworks like ARES [147] and TruLens[11] also provide similar LLM-based metrics, they could yield different results. However, this difference is expected to be minimal, as all these methods have similar evaluation targets and aspects [54]. Furthermore, the use of `GPT-4o` and Cohere's `command-r-plus` models introduces additional *construct validity* considerations. We selected the latest models from OpenAI and Cohere for the development of DOCCHAMELEON. While older models could yield different results, we believe that the minimal performance differences between newer models on retrieval and generation tasks will have a negligible impact on the overall performance of API documentation augmentation.

As for the *external validity*, the binary classifiers in this study were fine-tuned using Stack Overflow questions specifically related to TensorFlow API documentation, which may limit their generalizability. If the domain (e.g., Java, Python, C++) or the framework (e.g., PyTorch, Scikit-learn) differs from TensorFlow, the classifiers' performance could vary, potentially leading to misclassifications. However, this impact is likely to be minimal since the common keywords (*documentation*, *docs*, *document*, *official*, *website*) found in TensorFlow documentation-related questions are also prevalent in other domains when discussing documentation. Additionally, because Stack Overflow questions are structurally similar [156] and our primary focus is on the intended response type, the number of misclassifications should not be significant. Creating a "fully correct" labeled dataset is inherently challenging, which introduces a potential *external validity* concern with our

[11] https://www.trulens.org/trulens_eval/getting_started/core_concepts/rag_triad/

manually annotated datasets used for fine-tuning the binary classifiers. We used a well-established closed card sorting [195] technique to annotate the datasets in Sections 3.4.1 and 3.4.2. The high Fleiss' Kappa values obtained during both annotation processes indicate strong agreement among the annotators. Furthermore, the high proportion of graduate students in our qualitative study may also introduce *external validity* concerns. However, the impact on the generalization of our user study results is minimal, as the graduate students involved are master's and PhD candidates who work as research assistants in the fields of software engineering and machine learning. Moreover, half of these graduate students have previous professional industry experience, and nearly one-fourth of the participants in our study are currently working in the industry.

In our study, we used a limited dataset of 32 TensorFlow API documentation-related questions with accepted answers (see Section 3.4.3), which may raise potential *conclusion validity* concerns. However, we believe the impact is minimal because the dataset consists of real-world issues on Stack Overflow, with accepted answers recognized as reliable in previous literature [191]. Additionally, our analysis shows that 92.30% of TensorFlow documentation-related Stack Overflow questions arise from documentation ambiguities and inadequate examples (see Section 3.2). Therefore, this dataset, which focuses on questions requiring explanations and code examples, effectively covers a broad range of relevant TensorFlow documentation cases. Moreover, the results of our user study may not always be entirely accurate, potentially introducing *conclusion validity* concerns, as developers do not consistently provide a comprehensive perspective of real-world practices [98, 45]. However, user studies have been widely recognized as an effective research tool in software engineering.

In our user study, there is a possibility that some participants may have interpreted the questionnaire differently or found it unclear, potentially introducing concerns related to *internal validity*. To mitigate this risk, the questionnaire was designed iteratively. The initial draft was tested with a small group, and subsequent versions were reviewed and tested to ensure clarity and coverage of all aspects related to the parts added by

DOCCHAMELEON to augment the API documentation, which were evaluated by an independent party. Additionally, we developed an evaluation portal (see Section 3.6.4) to facilitate the user study. However, some users might have found the portal challenging to navigate, introducing another potential *internal validity* concern. We addressed this by initially testing the portal with a small group of four people and using their feedback to improve its usability, thereby making the evaluation process more user-friendly and accessible.

3.10 Conclusion

API documentation has traditionally been augmented without addressing documentation-related issues. Our analysis of TensorFlow documentation-related issues reveals that 92.55% are due to ambiguities, inadequate examples, and requests for additional resources. Additionally, we observe a growing trend in these issues on GitHub and Stack Overflow following major releases. This paper introduces DOCCHAMELEON, which augments TensorFlow API documentation using LLMs and the RAG method by generating explanations, code examples, and recommending related YouTube tutorials and Stack Overflow posts.

We compared DOCCHAMELEON against three baseline methods to evaluate the relevance of retrieved Stack Overflow information for addressing documentation-related questions. We then assessed the correctness, similarity, and faithfulness of DOCCHAMELEON generated explanations and code examples using ground truth answers. We conducted a user study to determine developers' perspectives on the relevance, accuracy, and helpfulness of the content added by DOCCHAMELEON, along with their preference between the official and augmented API documentation.

Our findings show that DOCCHAMELEON retrieves 30.35% more relevant information from Stack Overflow compared to baselines for addressing documentation-related questions, and generates explanations and code examples that are correct, similar, and faithful to the ground truth. Qualitative results also indicate that the DOCCHAMELEON

augmented content is relevant, accurate, and helpful, with 93.24% of participants preferring the augmented API documentation over the official version for programming tasks. We argue that generative AI can dynamically augment TensorFlow API documentation and potentially reduce documentation-related questions on Stack Overflow, addressing the growing trend following major TensorFlow releases.

Chapter 4

Conclusion and Future Work

The increased popularity of machine learning has attracted individuals from diverse fields. While these individuals are familiar with the application domains, they often lack experience in programming and software engineering. Our study, conducted at the intersection of machine learning and software engineering, shows the need for documentation to be tailored to different user skill levels due to the widespread adoption of machine learning. Moreover, developers have encountered issues when using current machine learning documentation, specifically TensorFlow, mostly due to inadequate examples and documentation ambiguities. These issues can make the documentation less useful, increase user questions, and force developers to seek help from third-party resources. Recent advancements in generative AI can be effectively utilized to enhance machine learning documentation by addressing documentation-related issues and recommending resources for background knowledge. This improvement increases the documentation's usefulness and makes it more accessible to a broader range of users.

4.1 Conclusion

In Chapter 2, we conducted a case study at the intersection of machine learning and software engineering. We mined Stack Overflow questions related to TensorFlow documentation. We constructed a comprehensive taxonomy to identify the nature and causes of issues encountered by developers using TensorFlow documentation. Additionally,

we analyzed TensorFlow tutorials, which uniquely recognize the need for tailored documentation based on user skill levels, evaluating their content for differences in readability and technicality. Our analysis extended to examining the profiles of users who referred to these tutorials within their questions. Our findings revealed that the majority of issues related to TensorFlow documentation stem from the inadequacy of provided examples. Furthermore, our quantitative study of TensorFlow tutorials showed no substantial overlap in topics between skill levels. Beginner tutorials contained notably more examples and higher readability scores, signifying ease of understanding. We also observed that all tutorials attracted questions from developers of varying skill levels. These analyses contribute to a better understanding of how documentation-related issues are discussed in Stack Overflow and emphasize the importance of tailoring documentation to accommodate a variety of skill levels.

Our findings revealed that addressing TensorFlow documentation-related questions concerning inadequate examples, ambiguities, and the need for additional resources resolves 92.55% of the questions posted on Stack Overflow. In Chapter 3, we introduced DocChameleon, a tool designed to augment TensorFlow API documentation by leveraging generative AI, specifically LLMs and the RAG technique. DocChameleon addresses documentation ambiguities by providing clear explanations and resolves the issue of inadequate examples by generating executable code examples. Additionally, it recommends YouTube tutorials and related Stack Overflow Q&As, fulfilling the demand for additional resources. Our analysis shows that DocChameleon generates explanations and code examples that are correct, similar, and faithful to ground truth answers. Furthermore, around 90% of user study participants found the explanations, code examples, Stack Overflow post links, and YouTube tutorial links included by DocChameleon in the API documentation to be relevant, accurate, and helpful. Additionally, 93.24% of participants preferred the augmented API documentation over the official documentation.

4.2 Future Work

In this thesis, we focused on software documentation at the intersection of machine learning and software engineering, with a specific focus on TensorFlow and Python. Currently, techniques for the automated generation and augmentation of software documentation produce a single version of the documentation for all users. Among various machine learning library documentation, only TensorFlow has recognized the need for tailored documentation to accommodate users of varying skill levels, introducing beginner and advanced level tutorials exclusively for Python. Our analysis demonstrated that these tutorials attract developers with diverse skill levels. Future research can extend this analysis to determine the optimal number of skill levels to consider in machine learning documentation and can explore the type of information that should be included based on these skill levels.

In our study, we underscore the importance of personalizing documentation to align with users' skill levels, emphasizing that one documentation does not fit all. Providing users with varying skill levels with a single version of the documentation can lead to an increase in questions on Q&A forums, and third-party tutorials and documentation. Future research can build on our findings by assessing users' skill levels based on their reputation, v-index, and experience using their Stack Overflow profiles. This information can be used to develop an automatic documentation customization technique employing a two-step process: first, identifying the users' skill levels, and second, tailoring the content to match these skill levels.

Our study introduced a tool to enhance machine learning API documentation by leveraging recent advancements in generative AI. We demonstrated that DocCHAMELEON can generate executable code examples to augment API documentation with a success rate of 56.25%. However, additional studies and benchmark datasets are needed to assess the functional correctness of the generated code for machine learning tasks by testing it against relevant test cases. Despite the increasing use of machine learning across various fields through programming, the existing literature has largely focused

on code generation within the software engineering domain [92, 38, 90, 123]. Future research could explore generating code examples specifically for machine learning tasks, enabling more comprehensive evaluations of generated code using test cases.

Bibliography

[1] *5 Levels of Text Splitting*. https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb. Accessed: Jun 2024.

[2] Nahla J Abid et al. "Using stereotypes in the automatic generation of natural language summaries for c++ methods". In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2015, pp. 561–565.

[3] L. A. Adamic et al. "Knowledge Sharing and Yahoo Answers: Everyone Knows Something". In: *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008.

[4] Emad Aghajani et al. "Software documentation issues unveiled". In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE. 2019, pp. 1199–1210.

[5] Emad Aghajani et al. "Software documentation issues unveiled". In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE. 2019, pp. 1199–1210.

[6] Emad Aghajani et al. "Software documentation: the practitioners' perspective". In: *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE. 2020, pp. 590–601.

[7] Arshad Ahmad et al. "A survey on mining stack overflow: question and answering (Q&A) community". In: *Data Technologies and Applications* 52.2 (2018), pp. 190–247.

[8] Mubashir Ali, Asad Aftab, and Wasi Haider Buttt. "Automatic Release Notes Generation". In: *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*. 2020, pp. 76–81. DOI: 10.1109/ICSESS49938.2020.9237671.

[9] Mubashir Ali, M Irtaza Nawaz Tarar, and Wasi Haider Butt. "Automatic release notes generation: a systematic literature review". In: *2020 IEEE 23rd International Multitopic Conference (INMIC)*. IEEE. 2020, pp. 1–5.

[10] M. Alshangiti et al. "Why is Developing Machine Learning Applications Challenging? A Study on Stack Overflow Posts". In: *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. Los Alamitos, CA, USA: IEEE Computer Society, 2019, pp. 1–11. DOI: 10.1109/ESEM.2019.8870187. URL: https://doi.ieeecomputersociety.org/10.1109/ESEM.2019.8870187.

[11] Moayad Alshangiti et al. "Why is developing machine learning applications challenging? a study on stack overflow posts". In: *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE. 2019, pp. 1–11.

[12] A. Anderson et al. "Discovering Value from Community Activity on Focused Question Answering Sites: A Case Study of StackOverflow". In: *KDD*. 2012.

[13] Giuliano Antoniol et al. "Assessing Staffing Needs for a Software Maintenance Project through Queuing Simulation". In: *IEEE Trans. Software Eng.* 30.1 (2004), pp. 43–58.

[14] John Anvik, Lyndon Hiew, and Gail C. Murphy. "Who should fix this bug?" In: *Proceedings of the 28th international conference on Software engineering – ICSE 2006*. 2006, pp. 361–370.

[15] Yuu Arimatsu et al. "Enriching API Documentation by Relevant API Methods Recommendation Based on Version History". In: *2018 IEEE Third International Workshop on Dynamic Software Documentation (DySDoc3)*. IEEE. 2018, pp. 15–16.

[16] Dorra Attiaoui, Arnaud Martin, and Boutheina Ben Yaghlane. “Belief measure of expertise for experts detection in question answering communities: Case study stack overflow”. In: *Procedia computer science* 112 (2017), pp. 622–631.

[17] Stephen H Bach et al. “Promptsource: An integrated development environment and repository for natural language prompts”. In: *arXiv preprint arXiv:2202.01279* (2022).

[18] Elma Bajraktari, Thomas Krause, and Christian Kücherer. “Documentation of Non-Functional Requirements for Systems with Machine Learning Components”. In: *Proceedings http://ceur-ws. org ISSN* 1613 (2024), p. 0073.

[19] Sebastian Baltes, Christoph Treude, and Martin P Robillard. “Contextual documentation referencing on stack overflow”. In: *IEEE Transactions on Software Engineering* 48.1 (2020), pp. 135–149.

[20] Jack Bandy and Nicholas Vincent. “Addressing" documentation debt" in machine learning research: A retrospective datasheet for bookcorpus”. In: *arXiv preprint arXiv:2105.05241* (2021).

[21] Thomas T Barker. “Writing software documentation”. In: *A Task-oriented Approach, Neddham* (1998).

[22] Anton Barua, Stephen W Thomas, and Ahmed E Hassan. “What are developers talking about? an analysis of topics and trends in stack overflow”. In: *Empirical software engineering* 19 (2014), pp. 619–654.

[23] Milos Bastajic, Jonatan Boman Karinen, and Jennifer Horkoff. “Operationalizing Machine Learning Using Requirements-Grounded MLOps”. In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer. 2024, pp. 231–248.

[24] Emily M Bender et al. “On the dangers of stochastic parrots: Can language models be too big?” In: *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 2021, pp. 610–623.

[25] Avinash Bhat et al. “Aspirations and practice of ml model documentation: Moving the needle with nudging and traceability”. In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 2023, pp. 1–17.

[26] Lutz Bornmann and Hans-Dieter Daniel. “What do we know about the h index?” In: *Journal of the American Society for Information Science and technology* 58.9 (2007), pp. 1381–1385.

[27] Amiangshu Bosu et al. “Building reputation in stackoverflow: an empirical investigation”. In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE. 2013, pp. 89–92.

[28] S. Budalakoti, D. DeAngelis, and K. S. Barber. “Expertise Modeling and Recommendation in Online Question and Answer Forums”. In: *Computational Science and Engineering, 2009. CSE’09. International Conference on. Vol. 4. IEEE*. 2009.

[29] *Build TensorFlow input pipelines*. https://www.tensorflow.org/guide/data#creating_an_iterator. Accessed: Nov 2023.

[30] Raymond PL Buse and Westley Weimer. “Synthesizing API usage examples”. In: *2012 34th International Conference on Software Engineering (ICSE)*. IEEE. 2012, pp. 782–792.

[31] Raymond PL Buse and Westley R Weimer. “Automatic documentation inference for exceptions”. In: *Proceedings of the 2008 international symposium on Software testing and analysis*. 2008, pp. 273–282.

[32] Raymond PL Buse and Westley R Weimer. “Automatically documenting program changes”. In: *Proceedings of the 25th IEEE/ACM international conference on automated software engineering*. 2010, pp. 33–42.

[33] Meng Cao et al. “Factual Error Correction for Abstractive Summarization Models”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2020, pp. 6251–6258. DOI: 10.18653/v1/2020.emnlp-main.506. URL: https://aclanthology.org/2020.emnlp-main.506.

[34] Yingkui Cao et al. “Refining traceability links between code and software documents”. In: *Proceedings of the 9th Asia-Pacific Symposium on Internetware*. 2017, pp. 1–10.

[35] Jiyoo Chang and Christine Custis. “Understanding implementation challenges in machine learning documentation”. In: *Proceedings of the 2nd ACM Conference on Equity and Access in Algorithms, Mechanisms, and Optimization*. ACM. 2022, pp. 1–8.

[36] Cong Chen and Kang Zhang. “Who asked what: Integrating crowdsourced faqs into api documentation”. In: *Companion Proceedings of the 36th International Conference on Software Engineering*. 2014, pp. 456–459.

[37] Jie-Cherng Chen and Sun-Jen Huang. “An empirical analysis of the impact of software development problem factors on software maintainability”. In: *Journal of Systems and Software* 82.6 (2009), pp. 981–992.

[38] Mark Chen et al. “Evaluating large language models trained on code”. In: *arXiv preprint arXiv:2107.03374* (2021).

[39] Xiang Cheng et al. “Exploiting user feedback for expert finding in community question answering”. In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE. 2015, pp. 295–302.

[40] Florin Cuconasu et al. “The power of noise: Redefining retrieval for rag systems”. In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2024, pp. 719–729.

[41] Liu D-R. et al. “Integrating expert profile, reputation and link analysis for expert finding in question-nswering websites”. In: *Proceedings of ACM Symposium on Applied Computing*. 2010.

[42] Barthélémy Dagenais and Martin P Robillard. “Using traceability links to recommend adaptive changes for documentation evolution”. In: *IEEE Transactions on Software Engineering* 40.11 (2014), pp. 1126–1146.

[43] Daniel Hasan Dalip et al. “Exploiting user feedback to learn to rank answers in q&a forums: a case study with stack overflow”. In: *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. 2013, pp. 543–552.

[44] Wei Ding et al. “Knowledge-based approaches in software documentation: A systematic literature review”. In: *Information and Software Technology* 56.6 (2014), pp. 545–567.

[45] Tore Dyba, Barbara A Kitchenham, and Magne Jorgensen. “Evidence-based software engineering for practitioners”. In: *IEEE software* 22.1 (2005), pp. 58–65.

[46] Brian P Eddy et al. “Evaluating source code summarization techniques: Replication and expansion”. In: *2013 21st International Conference on Program Comprehension (ICPC)*. IEEE. 2013, pp. 13–22.

[47] Mathias Ellmann et al. “Find, understand, and extend development screencasts on youtube”. In: *Proceedings of the 3rd ACM SIGSOFT International Workshop on Software Analytics*. 2017, pp. 1–7.

[48] Shahul Es et al. *RAGAS: Automated Evaluation of Retrieval Augmented Generation*. 2023. arXiv: 2309.15217 [cs.CL].

[49] Angela Fan et al. “Large language models for software engineering: Survey and open problems”. In: *arXiv preprint arXiv:2310.03533* (2023).

[50] Sen Fang et al. “Prhan: Automated pull request description generation based on hybrid attention network”. In: *Journal of Systems and Software* 185 (2022), p. 111160.

[51] Rudolf Flesch. “Flesch-Kincaid readability test”. In: *Retrieved October* 26.3 (2007), p. 2007.

[52] Andrew Forward and Timothy C Lethbridge. “The relevance of software documentation, tools and technologies: a survey”. In: *ACM symposium on Document engineering*. ACM. 2002, pp. 26–33.

[53] Daniel Fried et al. “Incoder: A generative model for code infilling and synthesis”. In: *arXiv preprint arXiv:2204.05999* (2022).

[54] Yunfan Gao et al. “Retrieval-augmented generation for large language models: A survey”. In: *arXiv preprint arXiv:2312.10997* (2023).

[55] Aidan Gomez. *Introducing Command R+: A Scalable LLM Built for Business*. https://cohere.com/blog/command-r-plus-microsoft-azure. Accessed: Aug 2024.

[56] Jean-Christophe Goulet-Pelletier and Denis Cousineau. “A review of effect sizes and their confidence intervals, Part I: The Cohen’sd family”. In: *The Quantitative Methods for Psychology* 14.4 (2018), pp. 242–265.

[57] Latifa Guerrouj, David Bourque, and Peter C Rigby. “Leveraging informal documentation to summarize classes and methods in context”. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 2. IEEE. 2015, pp. 639–642.

[58] JoAnn T. Hackos. “Online Documentation: The next Generation”. In: *Proceedings of the 15th Annual International Conference on Computer Documentation*. SIGDOC ’97. Salt Lake City, Utah, USA: Association for Computing Machinery, 1997, 99–104. ISBN: 0897918614. DOI: 10.1145/263367.263383. URL: https://doi.org/10.1145/263367.263383.

[59] Alaleh Hamidi et al. “Towards Understanding Developers’ Machine-Learning Challenges: A Multi-Language Study on Stack Overflow”. In: *21st IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2021, Luxembourg, September 27-28, 2021*. IEEE, 2021, pp. 58–69. DOI: 10.1109/SCAM52516.2021.00016. URL: https://doi.org/10.1109/SCAM52516.2021.00016.

[60] Gretchen Hargis. “Readability and computer documentation”. In: *ACM Journal of Computer Documentation (JCD)* 24.3 (2000), pp. 122–131.

[61] Yalda Hashemi, Maleknaz Nayebi, and Giuliano Antoniol. “Documentation of machine learning software”. In: *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2020, pp. 666–667.

[62] Hangfeng He, Hongming Zhang, and Dan Roth. “Rethinking with retrieval: Faithful large language model inference”. In: *arXiv preprint arXiv:2301.00303* (2022).

[63] Yichen He et al. “COME: Commit Message Generation with Modification Embedding”. In: *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2023, pp. 792–803.

[64] Gholamreza Hesamian. “One-way ANOVA based on interval information”. In: *International Journal of Systems Science* 47.11 (2016), pp. 2682–2690.

[65] Daniel Hoffman and Paul Strooper. “API documentation with executable examples”. In: *Journal of Systems and Software* 66.2 (2003), pp. 143–156.

[66] Xing Hu et al. “Practitioners’ expectations on automated code comment generation”. In: *Proceedings of the 44th International Conference on Software Engineering*. 2022, pp. 1693–1705.

[67] Xing Hu et al. “Practitioners’ Expectations on Automated Code Comment Generation”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, 1693–1705. ISBN: 9781450392211. DOI: 10.1145/3510003.3510152. URL: https://doi.org/10.1145/3510003.3510152.

[68] Moneeba Iftikhar, Sohail Riaz, and Zahid Yousaf. “Impact of YouTube Tutorials in Skill Development among University Students of Lahore.” In: *Pakistan Journal of Distance and Online Learning* 5.2 (2019), pp. 125–138.

[69] Ivana Clairine Irsan et al. “AutoPRTitle: A Tool for Automatic Pull Request Title Generation”. In: *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2022, pp. 454–458.

[70] Md Johirul Islam et al. “What Do Developers Ask About ML Libraries? A Large-scale Study Using Stack Overflow”. In: *arXiv preprint arXiv:1906.11940* (2019).

[71] Ziwei Ji et al. “Survey of hallucination in natural language generation”. In: *ACM Computing Surveys* 55.12 (2023), pp. 1–38.

[72] Nan Jia, Jie Chen, and Mingliang Li. “Encoding Beacon Statements for Code Comment Generation”. In: (2023).

[73] Huaxi Jiang et al. “DeepRelease: Language-agnostic Release Notes Generation from Pull Requests of Open-source Software”. In: *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. 2021, pp. 101–110.

[74] Siyuan Jiang, Ameer Armaly, and Collin McMillan. “Automatically generating commit messages from diffs using neural machine translation”. In: *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2017, pp. 135–146. DOI: 10.1109/ASE.2017.8115626.

[75] Siyuan Jiang and Collin McMillan. “Towards Automatic Generation of Short Summaries of Commits”. In: *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 2017, pp. 320–323. DOI: 10.1109/ICPC.2017.12.

[76] Meenu Mary John, Helena Holmström Olsson, and Jan Bosch. “Towards mlops: A framework and maturity model”. In: *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2021, pp. 1–8.

[77] Mira Kajko-Mattsson. “A survey of documentation practice within corrective maintenance”. In: *Empirical Software Engineering* 10 (2005), pp. 31–55.

[78] Hisashi Kamezawa et al. “Rnsum: A large-scale dataset for automatic release note generation via commit logs summarization”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022, pp. 8718–8735.

[79] Kristian Kersting et al. “Se4ml-software engineering for ai-ml-based systems (dagstuhl seminar 20091)”. In: (2020).

[80] Jinhan Kim et al. “Enriching Documents with Examples: A Corpus Mining Approach”. In: 31.1 (2013). ISSN: 1046-8188. DOI: 10.1145/2414782.2414783. URL: https://doi.org/10.1145/2414782.2414783.

[81] J Peter Kincaid et al. “Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel”. In: (1975).

[82] Takeshi Kojima et al. “Large language models are zero-shot reasoners”. In: *Advances in neural information processing systems* 35 (2022), pp. 22199–22213.

[83] B. Kou et al. “SOSum: A Dataset of Stack Overflow Post Summaries”. In: *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*. Los Alamitos, CA, USA: IEEE Computer Society, 2022, pp. 247–251. DOI: 10.1145/3524842.3528487. URL: https://doi.ieeecomputersociety.org/10.1145/3524842.3528487.

[84] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. “Machine learning operations (mlops): Overview, definition, and architecture”. In: *IEEE access* (2023).

[85] Shalom Lappin. “Assessing the strengths and weaknesses of Large Language Models”. In: *Journal of Logic, Language and Information* 33.1 (2024), pp. 9–20.

[86] Ralf Laue. “Anti-Patterns in End-User Documentation”. In: *Proceedings of the 22nd European Conference on Pattern Languages of Programs*. EuroPLoP ’17. Irsee, Germany: Association for Computing Machinery, 2017. ISBN: 9781450348485. DOI: 10.1145/3147704.3147726. URL: https://doi.org/10.1145/3147704.3147726.

[87] Huayang Li et al. “A survey on retrieval-augmented text generation”. In: *arXiv preprint arXiv:2202.01110* (2022).

[88] Jiarui Li, Ye Yuan, and Zehua Zhang. *Enhancing LLM Factual Accuracy with RAG to Counter Hallucinations: A Case Study on Domain-Specific Queries in Private Knowledge-Bases*. 2024. arXiv: 2403.10446 [cs.CL].

[89] Junyi Li et al. “HaluEval: A Large-Scale Hallucination Evaluation Benchmark for Large Language Models”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 6449–6464. DOI: 10.18653/v1/2023.emnlp-main.397. URL: https://aclanthology.org/2023.emnlp-main.397.

[90] Yujia Li et al. “Competition-level code generation with alphacode”. In: *Science* 378.6624 (2022), pp. 1092–1097.

[91] Jenny T Liang, Thomas Zimmermann, and Denae Ford. "Towards mining oss skills from github activity". In: *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. 2022, pp. 106–110.

[92] Jiawei Liu et al. "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation". In: *Advances in Neural Information Processing Systems* 36 (2024).

[93] Mingwei Liu et al. "Api-related developer information needs in stack overflow". In: *IEEE Transactions on Software Engineering* 48.11 (2021), pp. 4485–4500.

[94] Shangqing Liu et al. "ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking". In: *IEEE Transactions on Software Engineering* 48.5 (2022), pp. 1800–1817. DOI: 10.1109/TSE.2020.3038681.

[95] Yiheng Liu et al. "Summary of chatgpt-related research and perspective towards the future of large language models". In: *Meta-Radiology* (2023), p. 100017.

[96] Zhongxin Liu et al. "Automatic Generation of Pull Request Descriptions". In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2019, pp. 176–188. DOI: 10.1109/ASE.2019.00026.

[97] Zhongxin Liu et al. "Automating Just-In-Time Comment Updating". In: *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2020, pp. 585–597.

[98] David Lo, Nachiappan Nagappan, and Thomas Zimmermann. "How practitioners perceive the relevance of software engineering research". In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 2015, pp. 415–425.

[99] Ziyang Luo et al. "Wizardcoder: Empowering code large language models with evol-instruct". In: *arXiv preprint arXiv:2306.08568* (2023).

[100] Thomas W. MacFarland and Jan M. Yates. "Mann–Whitney U Test". In: *Introduction to Nonparametric Statistics for the Biological Sciences Using R*. Springer International Publishing, 2016, pp. 103–132. ISBN: 978-3-319-30634-6. DOI: 10.1007/978-3-319-30634-6_4. URL: https://doi.org/10.1007/978-3-319-30634-6_4.

[101] Laura MacLeod. "Reputation on Stack Exchange: Tag, You're It!" In: *2014 28th international conference on advanced information networking and applications workshops*. IEEE. 2014, pp. 670–674.

[102] Laura MacLeod. "Reputation on Stack Exchange: Tag, You're It!" In: *2014 28th International Conference on Advanced Information Networking and Applications Workshops*. 2014, pp. 670–674. DOI: 10.1109/WAINA.2014.108.

[103] Laura MacLeod, Margaret-Anne Storey, and Andreas Bergen. "Code, camera, action: How software developers document and share program knowledge using YouTube". In: *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE. 2015, pp. 104–114.

[104] Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. *SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models*. 2023. arXiv: 2303.08896 [cs.CL].

[105] Silverio Martínez-Fernández et al. "Software engineering for AI-based systems: a survey". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31.2 (2022), pp. 1–59.

[106] Richard E. Mayer. "Cognitive Theory of Multimedia Learning". In: *The Cambridge Handbook of Multimedia Learning*. Ed. by Richard E.Editor Mayer. Cambridge Handbooks in Psychology. Cambridge University Press, 2014, 43–71.

[107] Paul W. McBurney and Collin McMillan. "Automatic Documentation Generation via Source Code Summarization of Method Context". In: *Proceedings of the 22nd International Conference on Program Comprehension*. ICPC 2014. Hyderabad, India: Association for Computing Machinery, 2014, 279–290. ISBN: 9781450328791. DOI: 10.1145/2597008.2597149. URL: https://doi.org/10.1145/2597008.2597149.

[108] Paul W McBurney and Collin McMillan. "Automatic documentation generation via source code summarization of method context". In: *Proceedings of the 22nd International Conference on Program Comprehension*. 2014, pp. 279–290.

[109] Sarah Meldrum, Sherlock A Licorish, and Bastin Tony Roy Savarimuthu. "Crowdsourced knowledge on stack overflow: A systematic mapping study". In: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. 2017, pp. 180–185.

[110] Michael Meng, Stephanie M Steinhardt, and Andreas Schubert. "Optimizing API documentation: Some guidelines and effects". In: *Proceedings of the 38th ACM International Conference on Design of Communication*. 2020, pp. 1–11.

[111] D. M. Mimno and A. McCallum. "Expertise modeling for matching papers with reviewers." In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2007, pp. 500–509.

[112] Modal. *Custom containers*. https://modal.com/docs/guide/custom-container. Accessed: Aug 2024.

[113] João Eduardo Montandon et al. "Documenting apis with examples: Lessons learned with the apiminer platform". In: *2013 20th working conference on reverse engineering (WCRE)*. IEEE. 2013, pp. 401–408.

[114] Laura Moreno et al. "ARENA: an approach for the automated generation of release notes". In: *IEEE Transactions on Software Engineering* 43.2 (2016), pp. 106–127.

[115] Laura Moreno et al. "ARENA: An Approach for the Automated Generation of Release Notes". In: *IEEE Transactions on Software Engineering* 43.2 (2017), pp. 106–127. DOI: 10.1109/TSE.2016.2591536.

[116] Laura Moreno et al. "Automatic generation of natural language summaries for java classes". In: *2013 21st International conference on program comprehension (ICPC)*. IEEE. 2013, pp. 23–32.

[117] Laura Moreno et al. “Automatic generation of release notes”. In: *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering*. 2014, pp. 484–495.

[118] Fangwen Mu et al. “Developer-intent driven code comment generation”. In: *ICSE*. IEEE. 2023, pp. 768–780.

[119] Gail C Murphy. “Beyond integrated development environments: adding context to software development”. In: *2019 IEEE/ACM 41st international conference on software engineering: new ideas and emerging results (ICSE-NIER)*. IEEE. 2019, pp. 73–76.

[120] Brad A Myers and Jeffrey Stylos. “Improving API usability”. In: *Communications of the ACM* 59.6 (2016), pp. 62–69.

[121] AmirHossein Naghshzan and Sylvie Ratte. *Enhancing API Documentation through BERTopic Modeling and Summarization*. 2023. arXiv: 2308.09070 [cs.SE].

[122] Sristy Sumana Nath and Banani Roy. “Automatically generating release notes with content classification models”. In: *International Journal of Software Engineering and Knowledge Engineering* 31.11n12 (2021), pp. 1721–1740.

[123] Erik Nijkamp et al. “Codegen: An open large language model for code with multi-turn program synthesis”. In: *arXiv preprint arXiv:2203.13474* (2022).

[124] Kristian Nybom, Adnan Ashraf, and Ivan Porres. “A systematic mapping study on API documentation generation approaches”. In: *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2018, pp. 462–469.

[125] Ipek Ozkaya. “The next frontier in software development: AI-augmented software development processes”. In: *IEEE Software* 40.4 (2023), pp. 4–9.

[126] David Lorge Parnas. “Document based rational software development”. In: *Knowledge-Based Systems* 22.3 (2009), pp. 132–141.

[127] Chris Parnin and Christoph Treude. “Measuring API documentation on the web”. In: *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*. 2011, pp. 25–30.

[128] Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. “Seahawk: Stack overflow in the ide”. In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE. 2013, pp. 1295–1298.

[129] Luca Ponzanelli et al. “Too long; didn’t watch! extracting relevant fragments from software development video tutorials”. In: *Proceedings of the 38th international conference on software engineering*. 2016, pp. 261–272.

[130] Ruben Prieto-Diaz. “Status report: Software reusability”. In: *IEEE software* 10.3 (1993), pp. 61–66.

[131] *Python*. Accessed: Jun 2024. URL: https://peps.python.org/pep-0257/.

[132] Md. Mainur Rahman, Guenther Ruhe, and Thomas Zimmermann. “Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects”. In: *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. 2009, pp. 439–442. DOI: 10.1109/ESEM.2009.5316025.

[133] N. Raj, L. Dey, and B. Gaonkar. “Expertise Prediction for Social Network Platforms to Encourage Knowledge Sharing”. In: *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, 1*. 2011, pp. 380–383.

[134] Sebastian Raschka, Joshua Patterson, and Corey Nolet. “Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence”. In: *Information* 11.4 (2020), p. 193.

[135] Vikas Raunak, Arul Menezes, and Marcin Junczys-Dowmunt. “The Curious Case of Hallucinations in Neural Machine Translation”. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2021, pp. 1172–1183. DOI: 10.18653/v1/2021.naacl-main.92. URL: https://aclanthology.org/2021.naacl-main.92.

[136] Matthew Renze and Erhan Guven. "The Effect of Sampling Temperature on Problem Solving in Large Language Models". In: *arXiv preprint arXiv:2402.05201* (2024).

[137] Tal Ridnik, Dedy Kredo, and Itamar Friedman. "Code generation with alphacodium: From prompt engineering to flow engineering". In: *arXiv preprint arXiv:2401.08500* (2024).

[138] Martin P Robillard. "What makes APIs hard to learn? Answers from developers". In: *IEEE software* 26.6 (2009), pp. 27–34.

[139] Martin P. Robillard et al. "Automated API Property Inference Techniques". In: *IEEE Transactions on Software Engineering* 39.5 (2013), pp. 613–637. DOI: 10.1109/TSE.2012.63.

[140] Martin P. Robillard et al. "On-demand Developer Documentation". In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2017, pp. 479–483. DOI: 10.1109/ICSME.2017.17.

[141] Adriano M. Rocha and Marcelo A. Maia. "Automated API Documentation with Tutorials Generated From Stack Overflow". In: *Proceedings of the XXX Brazilian Symposium on Software Engineering*. SBES '16. Maringá, Brazil: Association for Computing Machinery, 2016, 33–42. ISBN: 9781450342018. DOI: 10.1145/2973839.2973847. URL: https://doi.org/10.1145/2973839.2973847.

[142] Adriano M Rocha and Marcelo A Maia. "Automated API documentation with tutorials generated from Stack Overflow". In: *Proceedings of the XXX Brazilian Symposium on Software Engineering*. 2016, pp. 33–42.

[143] Tobias Roehm. "Two user perspectives in program comprehension: end users and developer users". In: *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE. 2015, pp. 129–139.

[144] Negar Rostamzadeh et al. "Healthsheet: development of a transparency artifact for health datasets". In: *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*. 2022, pp. 1943–1961.

[145] Riccardo Rubei et al. “PostFinder: Mining Stack Overflow posts to support software developers”. In: *Information and Software Technology* 127 (2020), p. 106367.

[146] Vicinanza S., Mukhopadhyay T., and Prietula M. “Software Effort Estimation: An Exploratory Study of Expert Performance”. In: *Information Systems Research* 2 (1991), pp. 243–262.

[147] Jon Saad-Falcon et al. “Ares: An automated evaluation framework for retrieval-augmented generation systems”. In: *arXiv preprint arXiv:2311.09476* (2023).

[148] Lothar Sachs. *Applied statistics: a handbook of techniques*. Springer Science & Business Media, 2012.

[149] Marius Schlegel and Kai-Uwe Sattler. “Management of machine learning lifecycle artifacts: A survey”. In: vol. 51. 4. ACM New York, NY, USA, 2023, pp. 18–35.

[150] Xinyue Shen et al. “In chatgpt we trust? measuring and characterizing the reliability of chatgpt”. In: *arXiv preprint arXiv:2304.08979* (2023).

[151] Rodrigo Fernandes Gomes da Silva et al. “CROKAGE: effective solution recommendation for programming tasks by leveraging crowd knowledge”. In: *Empirical Software Engineering* 25 (2020), pp. 4707–4758.

[152] Diomidis Spinellis. “Code documentation”. In: *IEEE software* 27.4 (2010), pp. 18–19.

[153] Giriprasad Sridhara et al. “Towards automatically generating summary comments for java methods”. In: *Proceedings of the 25th IEEE/ACM international conference on Automated software engineering*. 2010, pp. 43–52.

[154] Jeffrey Stylos, Brad A Myers, and Zizhuang Yang. “Jadeite: improving API documentation using usage information”. In: *CHI'09 Extended Abstracts on Human Factors in Computing Systems*. 2009, pp. 4429–4434.

[155] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. “Live API documentation”. In: *Proceedings of the 36th international conference on software engineering*. 2014, pp. 643–652.

[156] Prissadang Suta et al. “Matching question and answer using similarity: an experiment with stack overflow”. In: *2018 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*. IEEE. 2018, pp. 51–54.

[157] *TensorFlow*. Accessed: Jun 2024. URL: https://www.tensorflow.org/community/contribute/docs.

[158] *TensorFlow: An Open Source Machine Learning Framework for Everyone*. https://www.tensorflow.org/. Accessed: Nov 2023.

[159] *TensorFlow: An Open Source Machine Learning Framework for Everyone*. https://www.TensorFlow.org/tutorial. Accessed: Nov 2023.

[160] Kyle Thayer, Sarah E Chasins, and Amy J Ko. “A theory of robust API knowledge”. In: *ACM Transactions on Computing Education (TOCE)* 21.1 (2021), pp. 1–32.

[161] Bill Thomas and Scott Tilley. “Documentation for software engineers: what is needed to aid system understanding?” In: *Proceedings of the 19th annual international conference on Computer documentation*. 2001, pp. 235–236.

[162] Lionel Nganyewou Tidjon et al. “An Empirical Study of Library Usage and Dependency in Deep Learning Frameworks”. In: *arXiv preprint arXiv:2211.15733* (2022).

[163] Oguzhan Topsakal and Tahir Cetin Akinci. “Creating large language model applications utilizing langchain: A primer on developing llm apps fast”. In: *International Conference on Applied Engineering and Natural Sciences*. Vol. 1. 1. 2023, pp. 1050–1056.

[164] Christoph Treude, Justin Middleton, and Thushari Atapattu. “Beyond accuracy: Assessing software documentation quality”. In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2020, pp. 1509–1512.

[165] Christoph Treude and Martin P. Robillard. "Augmenting API Documentation with Insights from Stack Overflow". In: ICSE '16. Austin, Texas: Association for Computing Machinery, 2016, 392–403. ISBN: 9781450339001. DOI: 10.1145/2884781.2884800. URL: https://doi.org/10.1145/2884781.2884800.

[166] Christoph Treude and Martin P Robillard. "Augmenting api documentation with insights from stack overflow. In 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)". In: *IEEE, 392ś403* (2016).

[167] Lewis Tunstall et al. "Efficient few-shot learning without prompts". In: *arXiv preprint arXiv:2209.11055* (2022).

[168] Gias Uddin, Foutse Khomh, and Chanchal K Roy. "Automatic api usage scenario documentation from technical q&a sites". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30.3 (2021), pp. 1–45.

[169] Gias Uddin, Foutse Khomh, and Chanchal K. Roy. "Automatic API Usage Scenario Documentation from Technical QA Sites". In: *ACM Trans. Softw. Eng. Methodol.* 30.3 (2021). ISSN: 1049-331X. DOI: 10.1145/3439769. URL: https://doi.org/10.1145/3439769.

[170] Gias Uddin and Martin P Robillard. "How API documentation fails". In: *Ieee software* 32.4 (2015), pp. 68–75.

[171] Shivani Upadhyay, Ehsan Kamalloo, and Jimmy Lin. "LLMs Can Patch Up Missing Relevance Judgments in Evaluation". In: *arXiv preprint arXiv:2405.04727* (2024).

[172] Sri Lakshmi Vadlamani and Olga Baysal. "Studying software developer expertise and contributions in Stack Overflow and GitHub". In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2020, pp. 312–323.

[173] Akhila Sri Manasa Venigalla and Sridhar Chimalakonda. "Understanding Emotions of Developer Community Towards Software Documentation". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. 2021, pp. 87–91. DOI: 10.1109/ICSE-SEIS52602.2021.00018.

[174] Thanh Trong Vu, Thanh-Dat Do, and Hieu Dinh Vo. "Context-Encoded Code Change Representation for Automated Commit Message Generation". In: *arXiv preprint arXiv:2306.14418* (2023).

[175] Haifeng Wang et al. "Pre-trained language models and their applications". In: *Engineering* (2022).

[176] Haoye Wang et al. "Context-Aware Retrieval-Based Deep Commit Message Generation". In: *ACM Trans. Softw. Eng. Methodol.* 30.4 (2021). ISSN: 1049-331X. DOI: 10.1145/3464689. URL: https://doi.org/10.1145/3464689.

[177] Rui Wang et al. "Role Prompting Guided Domain Adaptation with General Capability Preserve for Large Language Models". In: *arXiv preprint arXiv:2403.02756* (2024).

[178] Shaowei Wang et al. "Is reputation on Stack Overflow always a good indicator for users' expertise? No!" In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2021, pp. 614–618.

[179] Zora Zhiruo Wang et al. "CodeRAG-Bench: Can Retrieval Augment Code Generation?" In: *arXiv preprint arXiv:2406.14497* (2024).

[180] Stephen John Warnett and Uwe Zdun. "On the Understandability of MLOps System Architectures". In: vol. 50. 5. IEEE, 2024, pp. 1015–1039.

[181] Claes Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[182] Edmund Wong, Taiyue Liu, and Lin Tan. "Clocom: Mining existing source code for automatic comment generation". In: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE. 2015, pp. 380–389.

[183] Edmund Wong, Jinqiu Yang, and Lin Tan. "Autocomment: Mining question and answer sites for automatic comment generation". In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2013, pp. 562–567.

[184] D. Wu et al. “Leveraging Stack Overflow to Detect Relevant Tutorial Fragments of APIs”. In: *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Los Alamitos, CA, USA: IEEE Computer Society, 2021, pp. 119–130. DOI: 10.1109/SANER50967.2021.00020.

[185] Di Wu et al. “Leveraging stack overflow to detect relevant tutorial fragments of apis”. In: *Empirical Software Engineering* 28.1 (2023), p. 12.

[186] Di Wu et al. “Retrieving API knowledge from tutorials and stack overflow based on natural language queries”. In: *ACM Transactions on Software Engineering and Methodology* 32.5 (2023), pp. 1–36.

[187] Ye-Chi Wu, Lee Wei Mar, and Hewijin Christine Jiau. “CoDocent: Support API usage with code example and API documentation”. In: *2010 Fifth International Conference on Software Engineering Advances*. IEEE. 2010, pp. 135–140.

[188] Baoguo Yang and Suresh Manandhar. “Exploring user expertise and descriptive ability in community question answering”. In: *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*. IEEE. 2014, pp. 320–327.

[189] Ann Yuan et al. “Wordcraft: story writing with large language models”. In: *27th International Conference on Intelligent User Interfaces*. 2022, pp. 841–852.

[190] Jingxuan Zhang et al. “Enriching API documentation with code samples and usage scenarios from crowd knowledge”. In: *IEEE Transactions on Software Engineering* 47.6 (2019), pp. 1299–1314.

[191] Zhang Zhang et al. “An empirical study on the influence of social interactions for the acceptance of answers in Stack Overflow”. In: *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE. 2020, pp. 425–434.

[192] Haiyan Zhao et al. “Explainability for large language models: A survey”. In: *ACM Transactions on Intelligent Systems and Technology* 15.2 (2024), pp. 1–38.

[193] Lianmin Zheng et al. “Judging llm-as-a-judge with mt-bench and chatbot arena”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[194] Yue Zhou, Yue Yu, and Bo Ding. "Towards mlops: A case study of ml pipeline platform". In: *2020 International conference on artificial intelligence and computer engineering (ICAICE)*. IEEE. 2020, pp. 494–500.

[195] Thomas Zimmermann. "Card-sorting: From text to themes". In: *Perspectives on data science for software engineering*. Elsevier, 2016, pp. 137–141.