Using code from ChatGPT: Finding patterns in the developers' interaction with ChatGPT

ABSTRACT ChatGPT can advise developers and provide code on how to fix bugs, add new features, refactor, reuse, and secure their code but currently, there is little knowledge about whether the developers trust ChatGPT's responses and actually use the provided code. In this context, this study aims to identify patterns that describe the interaction of developers with ChatGPT with respect to the characteristics of the prompts and the actual use of the provided code by the developer. We performed a case study on 267,098 lines of code provided by ChatGPT related to commits, pull requests, files of code, and discussions between ChatGPT and developers. Our findings show that developers are more likely to integrate the given code snapshot in their code base when they have provided information to ChatGPT through several rounds of brief prompts that include problem-related specific words instead of using large textual or code prompts. Results also highlight the ability of ChatGPT to handle efficiently different types of problems across different programming languages.

Keywords: ChatGPT, bugs, features, reusing, developer interaction, association rules, case study, prompts, pattern concepts.

1 Introduction

The need for the timely release of defect-free software along with the accelerated development cycles places great pressure on software developers who often seek quick, high-quality solutions to the programming challenges that they confront [24]. The rise of ChatGPT [9] has introduced a new dimension to the developer community's quest for answers and solutions by accumulating and synthesizing knowledge from various sources, offering instantaneous and personalized assistance.

Currently, ChatGPT has emerged as a game-changer in the developer landscape, leveraging the power of Natural Language Processing to generate text that aids in multipurpose tasks. Even though it was not primarily introduced for software engineering, it seems to be able to provide advice on how to fix bugs, add new features, improve already-existent code, or even suggest tools and libraries to be reused for a specific purpose [25]. ChatGPT undoubtedly has disrupted the traditional paradigm of seeking programming help, mainly due to its judgment-free environment that provides fast responses. However, the question that arises is how relevant and accurate these responses are and whether developers trust and utilize the proposed code.

Up to now, there is little knowledge about how the developers use ChatGPT and whether the responses provided are relevant. This study attempts to shed light on the primary concerns raised by software developers in ChatGPT conversations, with respect to code acquisition, and the context under which the code provided is actually used. This involves analyzing the nature of concerns (programming language prompted,

tokens provided) and potential challenges (types of development tasks prompted), along with the developers' code base (GitHub repositories). The goal of the study is to identify frequently appearing patterns, in the form of association rules, that describe the correlation between prompt characteristics and the utilization (or not) of the generated code snippet. On this end, the study pursues two key goals:

- The first goal is to identify the main concerns of the developers when interacting with ChatGPT. For this purpose, we identify patterns that reflect the main issues presented by developers to ChatGPT along with the associated programming languages, and the type of development task in which they seek assistance. Through this first goal, the research will provide insights into common prompting patterns that reflect the needs of the developers' community that are expected to be addressed by ChatGPT.
- The second goal explores the various ways developers engage with ChatGPT and how these interaction styles influence the use (or not) of code from generated responses. The analysis will highlight whether developers primarily provide detailed instructions or adopt a more conversational approach. Furthermore, the research investigates whether a correlation exists between interaction styles and code utilization, offering recommendations on how developers should tailor their interactions to maximize the effectiveness of ChatGPT's contribution.

To address these inquiries, we conducted a case study utilizing DevGPT, a dataset comprising developer-ChatGPT conversations, including prompts and responses encompassing code snippets and associated GitHub repositories. Specifically, our analysis focused only on conversations where ChatGPT provided source code. We analyzed conversations concerning GitHub commits, GitHub pull requests, GitHub discussions, raw code files, and Hacker news posts. As a next step, we examined whether the code provided by ChatGPT was actually used by the developer, by checking the similarity of the provided code snippet in the project's GitHub repository. Then we searched for frequently appearing patterns in this context, aiming to identify the context under which developers use ChatGPT's generated code.

The rest of the paper is organized as follows: Section 2 summarizes the key contributions of previous research related to our topic of interest. Section 3 outlines the design of our case study, including the methodologies of data collection and analysis. Section 4 presents the results, organized by research question while in Section 5, we provide a discussion on the meaning and significance of the findings, we also address any potential limitations or threats to the validity of our study. Finally, in Section 6, we conclude the paper.

2 Related Work

Large-scale language models (LLMs) [20] are rapidly being adopted by software developers and applied to generate code and other artifacts associated with software engineering. Popular examples of LLM-based tools applied for these purposes include ChatGPT [4] and GitHub Copilot [12]

Discussions on LLM adoption in software engineering processes have centered on automated code generation and the security and code quality risks associated with the generated code. For example, Asare et al. [3] compared LLM code generation to humans from a security perspective. Similar research [7] has examined the quality of LLM-generated answers and code and LLM interaction patterns for fixing bugs [22]. The majority of studies found in the literature focus mainly on the ability of LLMS to fix programming bugs or focus on the security aspect of produced code. Surameery and Shakor [23] highlighted the potential of ChatGPT as a comprehensive debugging toolkit but stressed the need to be used supplementary with other debugging tools. Sobania et. al.[21] used a range of bug scenarios to test ChatGPT's efficacy in fixing bugs and concluded that it is competitive with the other automated tools while the interactive dialog window offered by ChatGPT increases the bug fix success rate [21].

In a similar direction, Jalil et. al. [15], examined in an educational environment, the replies of ChatGPT in BUG FIX scenarios and found that 44% of them are correct or partially correct, while the explanations provided are correct or partially correct in 57% of the cases. Other studies focused on ChatGPT's efficiency in providing answers to software architecture problems [2] and coding for various domains such as: in the field of 3D printing [3], in creating new general intelligence bots [18], and in typical numerical problems [13].

In all these studies the main conclusion is that ChatGPT when fed with suitable prompts can generate in several cases the relevant code leading to significant time saving. Furthermore, two studies suggest the use of prompt patterns to increase the probability of receiving an appropriate answer from ChatGPT [27]. White et.al., in [27]used a catalog of fast patterns and concluded that the proposed catalog improves by 4% the performance of ChatGPT. Similarly, in [26]the authors suggested a set of prompt patterns for tasks related to code quality improvement, refactoring, requirements elicitation, and software design.

Our research draws inspiration from these explorations and documents specific patterns that can be used to generate the desired code outcome from LLM models. The main contribution of this study is that, to our knowledge, this is the first study that examines the use of ChatGPT, and the appropriateness of the provided responses, in real-life multi-purpose code prompts that are associated with pre-existing projects hosted in GitHub. The access to the associated GitHub repository gives us the chance to study the level of originality of ChatGPT's results and verify whether the code included in the answer of ChatGPT was actually reused by practitioners and to which extent.

3 Case Study Design

In this section, we present the study design, by providing information about the research questions, the dataset used, and the employed analysis. The case study was designed and reported based on the guidelines of Runeson et al [18]

3.1 Research Questions

As discussed in the introductory section, the main objective of the present study is to explore the capability of ChatGPT to useful code and to evaluate the level of trust developers place in the utilization of the generated code. To align our research methodology with the underlying motivations, we define the following research questions and objectives:

RQ1: What are the main concerns of the developers when interacting with ChatGPT?

This research question aims to identify the main issues that developers present to ChatGPT and to provide an overall overview considering their frequency, characteristics, and potential repercussions.

In order to do this, we analyze the prompts that developers provide to ChatGPT in an attempt to deliver an overview that includes their frequency, characteristics, and possible outcomes. We isolate repetitive keywords from the titles, that are used to describe the prompts and identify patterns that are indicative of the co-existence of certain tasks. The analysis provides six different categories of tasks, which are: *BUG FIX, ENERGY, NEW FEATURE, REFACTOR, SECURITY, and OTHER* tasks. With this classification, we want to conclude whether there are dominant types of requests when interacting with LLMs such as ChatGPT.

RQ2: Are there any patterns that describe the interaction of developers with ChatGPT that are associated with the actual use of the response?

The purpose of RQ2 is to investigate the factors that influence the adoption of code generated by ChatGPT. Focusing on prompts, questions, and inherent characteristics of the interaction process we are looking for patterns that help towards receiving a useful answer from ChatGPT. For this reason, we first identify the originality of the AI-generated code compared to the given input. Sections of the code that resemble the original query submitted to ChatGPT are filtered out. In this question, we check whether developers are relying on ChatGPT's answers. For this reason, we examine if the after filtered code provided as an answer by ChatGPT can be found unaltered in the project repository mentioned by the developer. This question aims to highlight developers' reliance on LLMs.

To answer the aforementioned questions, the process presented in **Fig. 1.** was applied. The outlined approach consists of four phases that are (i) data collection, (ii) data preprocessing, (iii) discretization of variables with continuous values, (iv) data analysis, and (v) extraction of results accompanied by discussion of the most important findings.

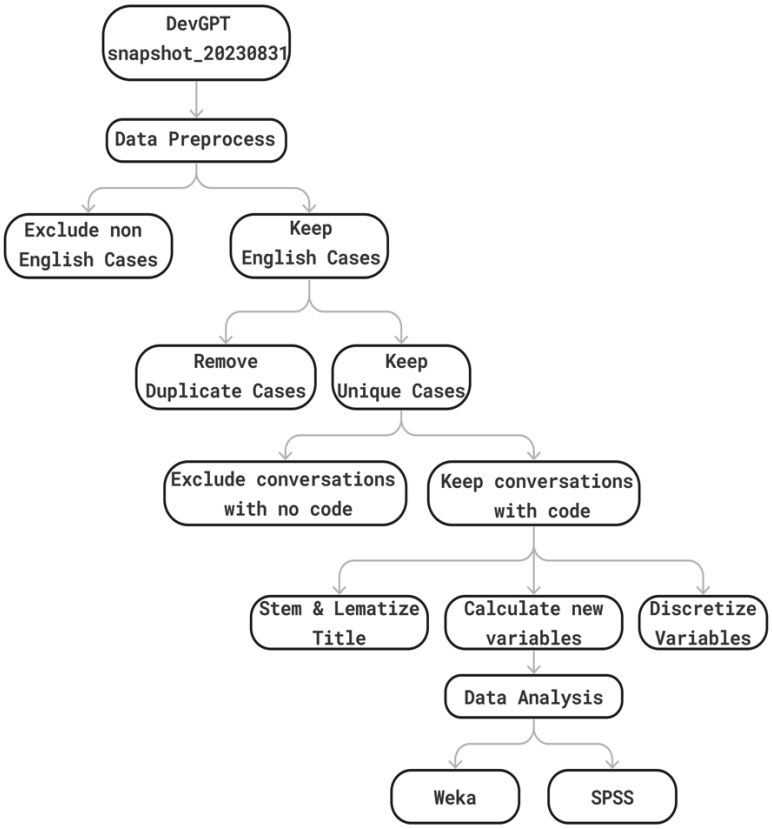

Fig. 1. Study map

3.2 Data Collection

The dataset utilized in this study is DevGPT [9], a curated dataset to explore how software developers interact with ChatGPT. The dataset is derived from shared ChatGPT conversations collected from GitHub and Hacker News. The moment this study took place, November 2023, the dataset encompassed 17,913 prompts and responses from ChatGPT, including 11,751 code snippets, that are linked to corresponding software development artifacts such—ranging from *source code, commits, issues, pull requests, to discussions and Hacker News threads*. The under-study snapshot, *snapshot_20230831*[1], consists of a total of 2,714 conversations between ChatGPT and users.

The DevGPT dataset is launched as a collection of six sets of JSON files. Each file includes the URL to the ChatGPT conversation[2], associated HTTP response status codes, access date of the URL, and the HTML response content and a link to the

[1] https://github.com/NAIST-SE/DevGPT/tree/main/snapshot_20230831
[2] https://help.openai.com/en/articles/7925741-chatgpt-shared-links-faq

developer's GitHub repository. Additionally, each conversation contains a list of prompts/answers, inclusive of any code snippets. The dataset provides details including the date of the conversation, the count of prompts/answers, their token information, and the model version involved in the chat.

From the enclosed information we kept the data that are related to: prompt title, number of prompts, the amount of used prompt (*NumberOfPrompts)* the number of tokens provided by ChatGPT's answers (*TokensOfAnswers*), the code snippets given by ChatGPT along with the number of the lines of code (*LinesGPT*) and links to the associated GitHub repositories, the programming language of the repository, the lines used by the developer in the aforementioned repository (*RepoLanguage*). We analyzed in total 2399 conversations out of which 379 commits ,138 pull requests, 347 issues, 40 discussions, 1305 code files, and 190 hacker news. The rest 315 cases were excluded.

3.3 Data Pre-Processing

The next step of the methodology involved the detection of the most useful fields to be analyzed, in accordance with the goals and objectives set by the proposed RQs. In respect to the RQs, the extracted features are presented in **Table 2.**

Initially, in the Data Pre-Processing part of our Analysis we implemented a script to handle the raw JSON files contained within the snapshot under investigation. and then we reviewed the processed data manually to identify and rectify any potential errors or discrepancies. As a next step we entered the data to Weka [9] to check their reliability and further analyze its insights.

The data **filtering process,** we followed a three-step process:

(a) We excluded all the cases where the language of user's prompt was different than English since keeping those cases would add uncertainty to our produced results.

(b) We removed all duplicate conversations. To determine a duplicate conversation, we checked all the attributes of the interaction including a) Link of conversation, b) Link of repository c) Individual prompts.

(c) We excluded all conversations where the Generative AI model did not include any code snippet in its answer.

After the filtering process the remain Dataset consisted of 2399 unique interactions. Regarding the **data preparation** the process included:

(a) Identification of tasks related to the prompts (*Type*): For this purpose, we extracted the separate words from the title of each prompt, with stemming techniques we removed associative and irrelevant words (and it, how, please, etc.) and through lemmatization we grouped words with the same root (i.e. add-> addition). As a second step we proceeded by classifying together words of similar context in six task categories using a keyword-based substring search method adopted from [1]. Additionally, we introduced a sixth category labeled *OTHER* to describe the cases that did not fall into any of the five categories yet led to code utilization.

In the end, each prompt was characterized as one of the following tasks: *BUG FIX, ENERGY awareness, NEW FEATURE, REFACTOR, SECURITY,* and *OTHER* software-related tasks.

(b) Calculation of the unique code lines provided by ChatGPT (*LinesGivenGPT)*: This variable indicates the number of unique lines of code provided by ChatGPT that the developer used. that were not included in the initial prompt of the developer. To calculate the value of the variable we first checked the code provided by the developer and compared this code with the code snippet provided by ChatGPT. Then we kept only the unique lines provided by ChatGPT (*LinesGivenGPT*)

(c) Calculation of the percentage of code lines, provided by ChatGPT, that were used by the developer (*LinesUsed)*: To calculate this variable we searched for the unique code lines provided by ChatGPT (*LinesGivenGPT*) within the GitHub repository linked to the associated conversation. In the case where part of the code was used, the additional variable *LinesUsed* was calculated as the number of lines of code provided by ChatGPT that were used by the developer to the total number of unique lines provided by ChatGPT.

The variables that participated in the study are presented in **Table 1** along with a description. All variables were extracted from the given DevGPT dataset and were later accordingly reformed though manual and automated processes.

Table 1. Variables Overview

Variable	Description
Type	Extracted type of JSON file (*FILE CODE, COMMIT, ISSUE, PULL REQUEST, DISCUSSION, and HACKER NEWS THREADS*)
RepoLanguage	The main language of the referred repository. For the purpose of our study we focused on (*HTML,PYTHON,JS,GO,C,BASH,JAVA*)
Type	Related to the question context (*BUG FIX, ENERGY, NEW FEATURE, REFACTOR, SECURITY, and OTHER*)
NumberOfPrompts	Number of prompts in this conversation
TokensOfPrompts	Number of tokens of prompts in this conversation
TokensOfAnswers	Tokens of answers in this conversation
Used CODE	Binary variable that states whether developer provided code.
LOC	Number of code lines provided by the developer.
CodeGPT	The main language of the answer assigned by ChatGPT.
LinesGPT	Code lines provided by ChatGPT.
LinesGivenGPT	Unique code lines provided by ChatGPT, after excluding code lines given by the user.
LinesUsed	Percentage of unique code lines provided by ChatGPT that were used by the developer

3.4 Discretization of variables with continuous values

Discretization of variables is used in data processing and involves partitioning the data range into intervals and assigning data points to their corresponding interval or categories. Discretization is essential for data mining processes as it facilitates the discovery of patterns, associations, and correlations. Equal frequency binning was chosen for this

purpose, as it offers clear insights into the distribution of observations and can handle outliers effectively [14]. The ranges defining the classes used for discretizing continuous variables are detailed in Section 3. This preprocessing step ensures the reliability of the association rule mining process, enabling meaningful insights to be derived from the data.

Based on the original range of variables, the following **Table 2** presents the 5 variables that were discretized along with the 5 distinct associated categories.

Table 2. Variable discretization in five classes

Variable	Very Low	Low	Average	High	Very High
NumberOfPrompts	≤2	3	4-8	9-20	21+
TokensOfPrompts	≤241	242-685	686-1094	1095- 3270	3271+
TokensOfAnswers	≤756	757-1584	1585-3745	3746-6505	6506+
LinesUsed	0	1-3	4-8	9-25	26+
LinesGivenGPT	≤2	2-8	9-21	22-39	40+

3.5 Data Analysis

The data analysis for this case study consists of three main sections. First, we estimate the frequency and descriptive statistics of the prompt characteristics. This involves identifying the frequency that specific features appear along with providing statistical summaries to better understand the general distribution of prompts the results Then we implemented statistical analysis to determine if there is a distinction in the adoption of the ChatGPT code depending on several prompt features. This test assists with identifying potential links between prompt features and generated code use patterns [14].

After that, we isolated frequently appearing item sets via visualization graphs and Association Rules (ARs). Association Rules are part of descriptive modeling strategies that seek to explain data and underlying relationships. This is achieved by establishing a set of rules that collectively define the variables of interest, shedding light on the associations and patterns within the dataset. Through these analyses, we get insights into the dynamics of prompt characteristics and the subsequent code adoption.

Given a set of observations over attributes A1, A2, …, An in a data set D a simple association rule has the following form:

$$A1 = X \text{ and } A2 = Y \Rightarrow A3 = Z$$

$$\text{Confidence} = P(A3 = Z \mid A1 = X, A2 = Y)$$

$$\text{Support} = \text{freq}(X \cap Y \cap Z, D).$$

This rule is interpreted as follows: when attribute A1 has the value X and attribute A2 has the value Y then there is a probability p (Confidence) that attribute A3 has the value Z. Confidence (C) is the probability p defined as the percentage of the records containing X, Y, and Z regarding the overall number of records containing X and Y only. Support (S) is a measure that expresses the frequency of the rule and is the ratio between the number of records that present X, Y, and Z to the total number of records in the data

set (D). In this study, we employed the Apriori algorithm to extract the Association Rules (ARs), with Weka [9].

4 Results

In this Section we present the results of this case study, organized by research question.
RQ1: What are the main concerns of the developers when interacting with ChatGPT?

To address the research question (RQ1) regarding the main concerns of developers when interacting with ChatGPT, we examined the distribution of conversation characteristics across different programming languages and types of conversation. **Table 3** provides insights into the distribution of characteristics such as the number of prompts, tokens of prompts, and lines used across various programming languages. Similarly, **Table 4** presents the distribution of conversation characteristics for different types of conversation, including *COMMIT, DISCUSSION, ISSUE, PULLREQUEST, HACKER NEWS, and CODE FILE.*

The distribution of conversation characteristics in **Table 3** and **Table 4** indicates that languages like *JAVA and JAVASCRIPT*, which exhibit higher percentages *of BUG FIX* tasks, also tend to have higher numbers of prompts and tokens, indicating potentially more complex or extensive interactions in these languages. While the rest of the programming languages (*RESTPL*) show lower percentages of BUG FIX tasks alongside lower numbers of prompts and tokens, suggesting a different usage pattern or focus.

Table 3. Distribution of conversation characteristics regarding the Programming Language

	BASH	C	HTML	JAVA	JS	NOPL	PYTHON	RESTPL
NumberOfPrompts	6.86	9.72	9.47	5.81	4.97	4.92	6.30	6.73
TokensOfPrompts	608.23	1594	1028	2383.31	635.61	700.97	1388.20	1091.45

Table 4. Distribution of conversation characteristics regarding the type of conversation

	COMMIT	DISCUSSION	ISSUE	PULLREQUEST	HACKER NEWS	CODE FILE
NumberOfPrompts	2.98	3.87	4.01	5.67	6.02	9.92
TokensOfPrompts	827.94	523.93	722.32	1736.03	708.46	1198.67

Analyzing **Table 5**, which depicts the distribution of tasks across different programming languages, we observe that tasks related to *REFACTOR* are prevalent across several languages, with the highest percentage observed in *BASH and HTML*. The *BUG FIX* task exhibits higher percentages in languages like *JAVA and JAVASCRIPT*

Table 5. Distribution of conversation characteristics regarding the type of conversation

	BASH	C	HTML	JAVA	JS	NOPL	PYTHON	RESTPL
SECURITY	2%	4%	41%	3%	**14%**	10%	19%	7%
BUG FIX	4%	11%	34%	**4%**	11%	14%	16%	5%
NEW FEATURE	2%	6%	33%	**4%**	**14%**	15%	**21%**	4%
REFACTOR	**5%**	8%	**53%**	2%	6%	9%	12%	5%
ENERGY	**5%**	10%	29%	0%	10%	**24%**	19%	5%
OTHER	3%	**12%**	32%	3%	12%	13%	17%	**9%**

In **Table 6** we examine the interconnected nature of developer concerns when interacting with ChatGPT.

In particular **Table 6** presents the occurrence of each type of task (% column), the percentage of the cases where the code provided by ChatGPT was utilized per each type of task (%Used column), and the pairwise frequency of tasks presented together in the same prompt.

REFACTOR tasks constitute the highest percentage of prompts (38.4%), indicating a prevalent need for code optimization and reuse among developers. Additionally, *BUG FIX and NEW FEATURE* implementation tasks are common, with moderate levels of utilization. *SECURITY*-related tasks, although less frequent, exhibit a relatively high utilization percentage, emphasizing their importance in ensuring the security of software systems. Lastly, a significant portion of prompts falls under the *OTHER* category, indicating a diverse range of developer needs beyond the primary task categories identified. The most frequent pair of tasks identified in the analysis is *NEW FEATURE REFACTOR*. This combination appears in 4.7% of the prompts and exhibits a utilization rate of 12.2%. This suggests that users seem to initiate their prompts by pinpointing specific programming tasks that require modification, due to new feature added bugs.

Table 6. Types of tasks prompted and the combinations of which led to code integration.

Tasks	%	%Uused	SECURITY	BUG FIX	NEW FEATURE	REFACTOR	ENERGY
SECURITY	5.1%	49.5%					
BUG FIX	20.6%	57.2%	2.8%				
NEW FEATURE	32.5%	57.2%	4.8%	2.2%			
REFACTOR	**38.4%**	**53.8%**	**4.7%**	**12.2%**	**23%**		
ENERGY	0.9%	66%	9%	57%	61%	38%	
OTHER	27.7%	53.2%	7.8%	1.8%	8.2%%	2%	0

Besides coupling keywords several cases included a combination with three or more keywords, making the prompts addressed to ChatGPT more specific. Among these the *NEW FEATURE REFACTOR BUG-FIX* is the most common triplet appearing in the

dataset with 41% of the time resulting in code adoption. As presented in **Table 7** when developers frequently seek advice from ChatGPT regarding security (*SECURITY*) concerns associated with newly added features (*NEW FEATURE*), often requesting guidance on refactoring (*REFACTOR*).

Table 7. Frequent task combination.

#	RULE	(S)	(C)
1	NEW FEATURE + SECURITY => REFACTOR	1.5%	63%
2	HTML => REFACTOR	43%	53%
3	NumberOfPrompts[VERY HIGH] => REFACTOR	16%	52%
4	PYTHON + REFACTOR => NEW FEATURE	4.7%	51%

RQ1: Developers mostly prompt 'REFACTOR' tasks, with the first ones receiving an answer that is used in most of the cases ('53.8%'). 'REFACTOR' prompts are commonly combined with other tasks to address concerns and needs. Developers' concerns vary depending on the prompted language.

RQ2: Are there any patterns that describe the interaction of developers with ChatGPT that are associated with the actual use of the response?

In this question, we check whether the answer of ChatGPT (code snippet) is used or not by the developer along with the level of usage (*LinesUsed*). We first produced the association map of **Fig. 2** which is a Relationship Graph between prompts' and answers' characteristics. The figure illustrates the connections between different combinations of values of the variables describing the prompts and the target variable, which is the *LinesUsed*. The edges in the model represent these connections, with their size indicating the frequency of the nodes and the thickness of the edges reflecting the strength of the association.

For instance, there is an association between the repository language HTML and the *Very Low* (≤2) number of prompts that is strong, i.e. presented in 201 instances. The same node led to *High and Very High* code adoption, hindering the need for specific keyword use rather than big explanatory texts to get the wanted result.

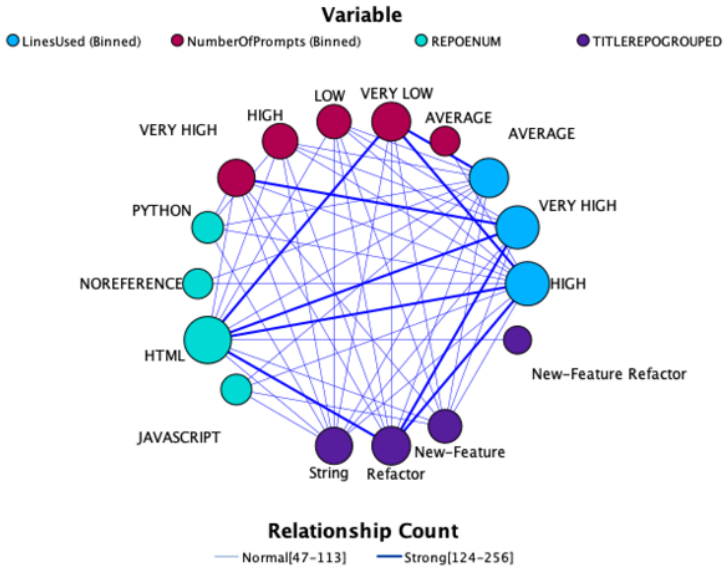

Fig. 2. Relationship Graph prompts', answers' characteristics that led to code utilization.

Additionally, to answer this RQ we identify frequently appearing patterns presented in **Table 8**

Table 8. Association Rules leading to code utilization.

#	RULE	(S)	(C)
1	COMMIT + NumberOfPrompts[LOW] => LinesUsed[VERY HIGH]	15.6%	73%
2	CODE FILE + PYTHON + NumberOfPrompts[AVERAGE] + OTHER => LinesUsed[HIGH]	3.7%	71%
3	COMMIT + NumberOfPrompts[LOW] + NEW FEATURE => LinesUsed[VERY HIGH]	7.3%	70%
4	LOC [AVERAGE] + NumberofTokens [LOW] + BUG FIX => LinesUsed[AVERAGE]	4.5%	78%
5	LOC [LOW] + NumberofTokens [LOW] + SECURITY => LinesUsed[HIGH]	5.3%	78%
6	CODE FILE+ NEW FEATURE + JAVA => LinesUsed[HIGH]	1.2%	65%
7	LOC [AVERAGE] + NumberofTokens [LOW] + BUG FIX => LinesUsed[AVERAGE]	7%	94%
8	COMMIT + NumberOfPrompts[VERY LOW] + REFACTOR => LinesUsed[AVERAGE]	17.5%	60%
9	CODE FILE + NumberOfPrompts[LOW] + BUG FIX=> LinesUsed[AVERAGE]	5.3%	55%
10	CODE FILE + BUG FIX => LinesUsed[AVERAGE]	11%	51%
11	HACKER NEWS + NumberOfPrompts[VERY LOW] + BUG FIX => LinesUsed[LOW]	18.3%	89%
12	HACKER NEWS + BUG FIX => LinesUsed[LOW]	34%%	65%
13	ISSUE + JAVASCRIPT + NEW FEATURE => LinesUsed[LOW]	9.8%	61%
14	CODE FILE + NEW FEATURE => LinesUsed[VERY LOW]	17.3%	63%
15	NO + NumberOfPrompts[VERY HIGH] => LinesUsed[NONE]	30%	92%
16	CODE FILE + REFACTOR => LinesUsed[NONE]	17.3%	84%

17	COMMIT +HTML => LinesUsed[NONE]	7.6%	74%
18	ISSUE + PYTHON => LinesUsed[NONE]	11%	50%
19	HTML+ NumberOfPrompts[LOW] + REFACTOR => LinesUsed[VERY HIGH]	9.8%	59%
20	JAVASCRIPT + NumberOfPrompts[AVERAGE] + NEW FEATURE => LinesUsed[VERY HIGH]	9.6%	53%
21	BASH + NumberOfPrompts[AVERAGE] + REFACTOR => LinesUsed[HIGH]	13%	77.3%
22	C + NumberOfPrompts[AVERAGE] => LinesUsed[HIGH]	16.8%	71%
23	HTML+ NumberOfPrompts[LOW] + REFACTOR => LinesUsed[VERY HIGH]	9.8%	59%
24	C + CODE FILE + REFACTOR => LinesUsed[HIGH]	22%	53%
25	HTML + NumberOfPrompts[VERY LOW] + REFACTOR => LinesUsed[AVERAGE]	16%%	58%
26	C + NEW FEATURE => LinesUsed[AVERAGE]	22%	53%
27	HTML+ NumberOfPrompts[VERY LOW] + BUG FIX => LinesUsed[AVERAGE]	11.3%	53%
28	PYTHON + NEW FEATURE + REFACTOR=> LinesUsed[NONE]	9.8%	92%
29	C + BUG FIX => LinesUsed[NONE]	17%	75%
30	BASH + NumberOfPrompts[AVERAGE] => LinesUsed[NONE]	29%	62%
31	NumberofTokens [HIGH] + LOC[LOW] => LinesUsed[NONE]	17%	73%

The rule number 19 is interpreted as following: the use of several rounds of prompts highly focusing on refactoring issues when referring to an HTML task results in a Very High level of code adoption.

Some general observations that stem from **Table 8** are that interactions involving HTML or JavaScript combined with 'REFACTOR' tasks tend to result in high code utilization. The characteristics of prompts significantly influence code adoption rates. Clarity (low/average number of prompts), specificity (code file provided), and relevance (mention a particular type of task for a specific programming language) contribute to higher levels of code utilization. A medium amount of back-and-forth interaction with GPT leads to better results than providing single long prompts or giving too much irrelevant info.

RQ2: Prompt characteristics influencing higher code adoption refer to specific tasks such as 'NEW FEATURE' and 'SECURITY' or interactions like 'COMMIT'. Developers should clearly state the general type of task expected in their prompts and ensure they are concise and specific. Developers should clearly state the general type of task that is expected as a response and use sufficient rounds of prompts as input. Long textual prompts that are not accompanied by code is not a good practice, but so is long parts of code without the appropriate explanation.

5 Discussion

In this research, we observed that users often interacted with ChatGPT in an informal, conversational manner, even when seeking specific code-related assistance as in xxx% of the cases the code provided by ChatGPT was not used. In particular, some observations that can be useful to practitioners are:

a. General textual input (high number of tokens) without being accompanied by technical information (code file, or adequate lines of code in the prompt) is insufficient. The informal interactions create a potential gap between user expectations and the system's response style.
b. Prompts related to small-scale programming tasks (Bug fixes, Security tasks) when accompanied with low or average lines of code are more likely to receive code that it is going to be used. It seems that small tasks and context-specific prompts are more efficiently handled by ChatGPT.
c. Programming/ descriptive languages that are loosely coupled such as JavaScript, HTML, and shell languages (BASH) tend to be frequently prompted and receive answers that are going to be used. In particular, REFACTORING and NEW FEATURE prompts in these cases are efficiently addressed.
d. Prompts that lack information in the text related to the Programming Language prompted or the specific type of task required also do not receive the code that is going to be used.

Based on these findings researchers can work on prompt engineering practices that are oriented on describing problems related to code generation. Prompt templates and good practices when prompting LLMs could increase the percentages of code utilization. Additionally, during our research, a large amount of ChatGPT's answer code was discarded as it was replicated or even copied from the same resource as the user's prompt. This raised a question on ethics, related to the source of ChatGPT-generated code, particularly concerning confidentiality and ownership matters. This issue should be researched and clarified prior to the formal integration of LLM's in the engineering lifecycle.

As a future work, we intend to explore how ChatGPT handles different types of issues or tasks specific to each programming language. By analyzing whether certain languages elicit more inquiries or if users consistently pose similar questions for each language, we can identify areas for improving model training. Moreover we are highly interested on investigating the resources and ethical considerations surrounding ChatGPT's code generation.

5.1 Threats to validity

This section presents the threats to the validity of the current research formulated according to Runeson's et al. [18] classification.

Regarding **Construct Validity**, we should mention that possible bias could be inserted in the model by the variable related to the type of task in which the prompt was classified based on the keywords of the title 1, in the case where the classification mechanism changes then the results might be different. In the future, we intend to use and compare the results with different classification schemas. Additionally, there is also bias in the calculation of the variable UsedCODE since we examine a particular GitHub repository time snapshot that differs across projects compared to the time when the prompt dialog took initially place. It is highly possible that the closer the dialog is to the snapshot we took the more likely is for the code to be used while as time passes the

code after being tested might have been removed. The “survival” of ChatGPT code in the developers’ repositories is also a subject of future research.

Internal validity, in this case, is not applicable since the examination of causal relationships is out of the scope of the study.

Concerning the **External validity** and in particular the generalizability supposition, changes in the responses might occur if the prompts analyzed were re-fed to ChatGPT, a fact that would alter the results as well. A future replication of this study, on the same or different prompts would be valuable.

Additionally, DevGPTs sample size is limited. As a result, we encourage future studies to replicate our findings using bigger sample sizes and alternative datasets. As our research on this topic is ongoing, we plan to address the threats by expanding our sample size and exploring the hypothesis in a more diverse set of human-ChatGPT interactions.

To increase the **Reliability** of the study, that reflects the reproducibility of study, we applied two mitigation actions: (a) we recorded the case study design protocoling in detail and (b) we uploaded the relevant tools that were used to obtain the data, along with the collected data in a GitHub repository.

6 Conclusions

ChatGPT offers developers guidance and code snippets for tasks like bug fixing, feature addition, and code security. The extent to which developers trust and utilize ChatGPT's responses remains unclear. Our study analyzed a dataset exceeding 267,098 lines of exchanged code. Interestingly, the research revealed that developers are more likely to integrate the provided code snippets when they use shorter, focused prompts that target specific problems over multiple interactions. Lengthy textual prompts or those that are already code-based seem to be less effective. Refactoring existing code or reusing existing solutions emerged as a prevalent task among developers, with a very high utilization rate (53.8%) for the provided responses from ChatGPT. Tasks addressing new features or security concerns, as well as interactions like commits, strongly influence the adoption of the provided code. To maximize effectiveness, our study suggests that developers should clearly articulate their tasks in concise prompts. Additionally, providing sufficient context and explanation alongside the code snippets for better understanding leads to better chances on code. Our study highlights the importance of tailored prompt strategies and clear communication in optimizing the utilization of ChatGPT’s provided code. Our study represents the beginning of exploring the dependent relationship between large language models like ChatGPT and developers. Based on our findings, we aim to conduct further investigations to deepen our understanding of this relationship.

REFERENCES

1. A. I. Champa, M. F. Rabbi, M. F. Zibran and M. R. Islam, "Insights into Female Contributions in Open-Source Projects," *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, Melbourne, Australia, 2023, pp. 357-361,

2. Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fahmideh, Mst Shamima Aktar, and Tommi Mikkonen. 2023. Towards Human-Bot Collaborative Software Architecting with ChatGPT. In Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering (EASE '23). Association for Computing Machinery, New York, NY, USA, 279–285. https://doi.org/10.1145/3593434.3593468
3. Asare Owura, Meiyappan Nagappan, and N. Asokan. 2023. Is GitHub's Copilot as bad as humans at introducing vulnerabilities in code? Empirical Softw. Engg. 28, 6 (Nov 2023). https://doi.org/10.1007/s10664-023-10380-1
4. Badini Silvia, Stefano Regondi, Emanuele Frontoni, Raffaele Pugliese,Assessing the capabilities of ChatGPT to improve additive manufacturing troubleshooting,Advanced Industrial and Engineering Polymer Research,Volume 6, Issue 3,2023,Pages 278-287,ISSN 2542-5048,https://doi.org/10.1016/j.aiepr.2023.03.003
5. Bang Y, Cahyawijaya S, Lee N, Dai W, Su D, Wilie B, Lovenia H, Ji Z, Yu T, Chung W, Do QV, Xu Y, Fung P (2023) A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity. arXiv:230204023
6. Biswas, S. (2023). Role of ChatGPT in Computer Programming. Mesopotamian Journal of Computer Science, 2023, 9–15. https://doi.org/10.58496/MJCSC/2023/002
7. Borji A (2023) A Categorical Archive of ChatGPT Failures. arXiv:230203494 [cs]
8. Daun Marian and Jennifer Brings. 2023. How ChatGPT Will Change Software Engineering Education. In Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023). Association for Computing Machinery, New York, NY, USA, 110–116. https://doi.org/10.1145/3587102.3588815
9. DevGPT: Studying Developer-ChatGPT Conversations: 2023. https://github.com/NAIST-SE/DevGPT. Accessed: 2023-12-11.
10. Etemadi Khashayar, Niloofar Tarighat, Siddharth Yadav, Matias Martinez, Martin Monperrus,Estimating the potential of program repair search spaces with commit analysis,Journal of Systems and Software,Volume 188,2022,111263,ISSN 0164-1212,https://doi.org/10.1016/j.jss.2022.111263
11. Frieder S, Pinchetti L, Griffiths R-R, Salvatori T, Lukasiewicz T, Petersen PC, Chevalier A, Berner J (2023) Mathematical Capabilities of ChatGPT. arXiv:230113867 [cs]
12. GitHub (2023) GitHub Copilot Your AI pair programmer. In: GitHub. https://github.com/features/copilot
13. Kashefi and Tapan Mukerji. 2023. ChatGPT for Programming Numerical Methods. *Journal of Machine Learning for Modeling and Computing* 4, 2 (2023), 1–74.
14. Witten, Ian & Hall, Mark & Frank, Eibe & Holmes, Geoffrey & Pfahringer, Bernhard & Reutemann, Peter. (2009). The WEKA data mining software: An update. SIGKDD Explorations. 11. 10-18. 10.1145/1656274.1656278.
15. Jalil, S. et al. 2023. ChatGPT and Software Testing Education: Promises & Perils. arXiv:2302.03287 [cs]. (Feb. 2023).
16. Peng, Liu & Qing, Wang & Yujia, Gu. (2009). Study on Comparison of Discretization Methods. Artificial Intelligence and Computational Intelligence, International Conference on. 4. 380-384. 10.1109/AICI.2009.385.
17. Owura Asare, Meiyappan Nagappan, and N. Asokan. 2023. Is GitHub's Copilot as bad as humans at introducing vulnerabilities in code? Empirical Softw. Engg. 28, 6 (Nov 2023). https://doi.org/10.1007/s10664-023-10380-1
18. Kathikar, Adhishree & Nair, Aishwarya & Lazarine, Ben & Sachdeva, Agrim & Samtani, Sagar. (2023). Assessing the Vulnerabilities of the Open-Source Artificial Intelligence (AI) Landscape: A Large-Scale Analysis of the Hugging Face Platform. 10.1109/ISI58743.2023.10297271.

19. Per Runeson, Höst M., Austen Rainer, and Björn Regnell, "Case Study Research in Software Engineering Guidelines and Examples." John Wiley & Sons, Inc, Hoboken, NJ, USA, 2012.
20. Rishi Bommasani, Hudson DA, Ehsan Adeli, Altman RB, Arora S, Sydney von Arx, Bernstein MS, Bohg J, Bosselut A, Brunskill E, Brynjolfsson E, Buch S, Card D, Castellon R, Chatterji NS, Chen AT, Creel K, Davis J, Demszky D, Donahue C (2021) On the Opportunities and Risks of Foundation Models. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2108.07258
21. Sobania, M. Briesch, C. Hanna and J. Petke, "An Analysis of the Automatic Bug Fixing Performance of ChatGPT," in 2023 IEEE/ACM International Workshop on Automated Program Repair (APR), Melbourne, Australia, 2023 pp. 23-30. doi: 10.1109/APR59189.2023.00012
22. S. M. Nasehi, J. Sillito, F. Maurer and C. Burns, "What makes a good code example?: A study of programming Q&A in StackOverflow," *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, Trento, Italy, 2012, pp. 25-34, doi: 10.1109/ICSM.2012.6405249.
23. Surameery, N.M.S. and Shakor, M.Y. 2023. Use Chat GPT to Solve Programming Bugs. International Journal of Information technology and Computer Engineering. 3, 31 (Jan. 2023), 17–22.
24. Xiao, Tao & Baltes, Sebastian & Hata, Hideaki & Treude, Christoph & Kula, Raula & Ishio, Takashi & Matsumoto, Kenichi. (2023). 18 Million Links in Commit Messages: Purpose, Evolution, and Decay.
25. Wahyu Rahmaniar 2023. ChatGPT for Software Development: Opportunities and Challenges. (Aug. 2023).
26. White, J. et al. 2023. ChatGPT Prompt Patterns for Improving Code Quality, REFACTORing, Requirements Elicitation, and Software Design. arXiv:2303.07839 [cs]. (Mar. 2023).
27. White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D.C. (2023). A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. ArXiv, abs/2302.11382.