Investigating Code Generation Performance of ChatGPT with Crowdsourcing Social Data
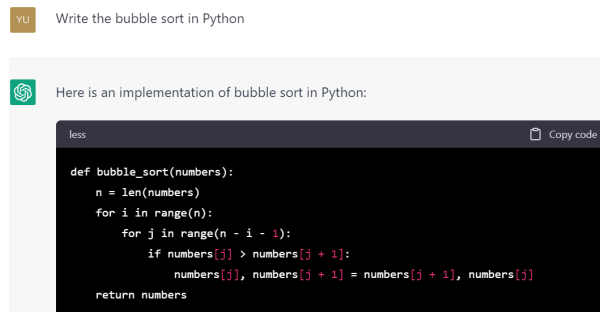
Abstract—The recent advancements in Artificial Intelligence, particularly in large language models and generative models, are reshaping the field of software engineering by enabling innovative ways of performing various tasks, such as programming, debugging, and testing. However, few existing works have thoroughly explored the potential of AI in code generation and software developers' attitudes toward AI-assisted coding tools. This knowledge gap leaves it unclear how AI is transforming software engineering and programming education. This paper presents a scalable crowdsourcing data-driven framework to investigate the code generation performance of generative large language models from diverse perspectives across multiple social media platforms. Specifically, we utilize ChatGPT, a popular generative large language model, as a representative example to reveal its insights and patterns in code generation. First, we propose a hybrid keyword word expansion method that integrates words suggested by topic modeling and expert knowledge to filter relevant social posts of interest on Twitter and Reddit. Then we collect 316K tweets and 3.2K Reddit posts about ChatGPT's code generation, spanning from Dec. 1, 2022 to January 31, 2023. Our data analytics show that ChatGPT has been used in more than 10 programming languages, with Python and JavaScript being the two most popular, for a diverse range of tasks such as code debugging, interview preparation, and academic assignment solving. Surprisingly, our analysis shows that fear is the dominant emotion associated with ChatGPT's code generation, overshadowing emotions of happiness, anger, surprise, and sadness. Furthermore, we construct a ChatGPT prompt and corresponding code dataset by analyzing the screenshots of ChatGPT code generation shared on social media. This dataset enables us to evaluate the quality of the generated code, and we will make the dataset available to the public soon. We believe the insights gained from our work will provide valuable guidance for future research on AI-powered code generation.

Index Terms—ChatGPT, Coding Generation, Software Engineering, Large Language Models (LLMs), Generative Models, Social Media

I. INTRODUCTION

Recently, the advancements in large language models (LLMs) and generative models have revolutionized many applications, including free text generation, question answering, and document summarization, enabling a wide range of real-world services such as AI robot lawyers [1] and AI music co-creation [2]. The field of coding, which involves writing tasks in certain programming languages, is also benefiting from the rapid development of generative LLMs. However, unlike traditional writing tasks, programming requires strict adherence to syntax and logic rules, making it more challenging for generative models to produce high-quality code.

Several studies have investigated the potential of LLMs in software development. For instance, Barke et al. [3] and Vaithilingam et al. [4] examined user perceptions of generative

Fig. 1. ChatGPT writes the bubble sort algorithm in Python

models in coding writing. However, many of these studies are based on case studies, with limited consideration of broader applications in software development. The emerging OpenAI's ChatGPT, a member of GPT-3 LLM family, demonstrates promising performance in code generation, attracting widespread attention from stakeholders in software engineering. As shown in Figure 1, ChatGPT can generate the bubble sort algorithm in Python with the prompt of "write the bubble sort in Python." Some studies have explored the use of ChatGPT for code generation tasks [5]–[7]. Nonetheless, these studies did not comprehensively evaluate the overall effectiveness of ChatGPT as a code generation and assistance tool on a large scale.

It is challenging to conduct a large-scale study on the performance of LLMs in code generation due to the following reasons. First, programming languages exhibit diverse syntax and are applicable to a wide range of tasks. For instance, SQL is primarily utilized in database operations, while JavaScript is commonly used in web programming. Second, code generation encompasses numerous programming tasks, including debugging, testing, and programming, for various stakeholders. Moreover, conducting user studies in the lab to investigate the code generation of LLMs can be costly and time-consuming. Therefore, conducting a comprehensive study on the performance of LLMs that covers numerous programming languages, tasks, and stakeholders poses significant challenges.

To address the aforementioned challenges, this paper proposes a scalable crowdsourcing data-driven framework that integrates multiple social media data sources to examine the code generation performance of ChatGPT. The proposed framework comprises three key components, namely keyword expansion, data collection, and data analytics. Specifically, we utilize topic modeling and expert knowledge to identify all

keywords that are relevant to programming in the context of ChatGPT, thus expanding the seed keyword of "ChatGPT." Using these expanded keywords, we retrieve 316K tweets and 3.2K Reddit posts related to ChatGPT's code generation from December 1, 2022, to January 31, 2023.

Furthermore, we conduct a comprehensive analysis using multimodal data (text and images) to answer the following research questions:

1) What are the most popular programming languages in ChatGPT usage?
2) What programming scenarios, tasks, and purposes are people using ChatGPT for?
3) What is the temporal distribution of the discussion on ChatGPT code generation?
4) How do stakeholders perceive ChatGPT code generation?
5) What are the prompts to generate code?
6) What is the quality of the code generated by ChatGPT?

To the best of our knowledge, the proposed work represents the first systematic study on emerging generative models in code writing and testing. In this paper, we summarize our contributions as follows:

- We have proposed a scalable crowdsourcing and social data-driven framework for investigating the code generation capabilities of ChatGPT.
- We have presented a novel hybrid keyword expansion method that incorporates words recommended by topic modeling and experts to ensure that most of the related social media posts are matched during data collection.
- Our study considers multiple social media platforms (Twitter and Reddit) and multimodal data (text and image) to mitigate potential biases caused by a single data source or data type.
- We have provided data analytics from multiple perspectives, including topic inference, sentiment analysis, and data quality measurement.
- We have built a real-world programming dataset containing the ChatGPT prompt and the associated generated code. This dataset will be released to the entire software engineering community soon.

II. RELATED WORK

As programming generation and assistant tools, such as CodeBERT [8] and IntelliCode Compose [9], become more widely used, there has been an increased focus on investigating the usability and interactions between users and code generation tools [3], [4], [10], [11]. For example, Barke et al. identified two interaction modes between programmers and code generation tools: acceleration mode and exploration mode, by observing how 20 programmers solved various tasks using the code generation tool Copilot [3]. Although this paper discussed a grounded theory of how programmers might interact differently with Copilot, it only investigated a limited number of users and did not reveal the reactions from users.

Vaithilingam et al. [4] performed a study on 24 participants consisting of different groups of people with minimal and moderate experience in using Copilot and IntelliSense. By quantitative and qualitative analysis, they observed that participants who used Copilot failed to complete tasks more. Unlike the above case study works, we investigate a large dataset of feedback from users of the code generation tool ChatGPT. Besides the user reactions, our study also examines the performance and limitations of ChatGPT.

As ChatGPT gains more attention recently, some researchers have studied its use for code generation [5]–[7]. Aljanabi et al. [5] discussed the possibility of using ChatGPT as a code generation tool. Avila et al. [6] have described how ChatGPT's programming capability can be used for developing online behavioral tasks, such as concurrent reinforcement schedules, in HTML, CSS, and JavaScript code. In their work, they created files with extensions .html, .css, and .js, which include the basic structure of the page, such as headings, linking with styling elements, and other dynamic elements. In contrast to the above works, we analyzed the performance of code generation using ChatGPT.

When it comes to complex coding, there is always a chance of unidentified bugs, which may lead to a code crash. Automated Program Repair (APR) is a concept introduced to provide automatic fixes for detected errors. In recent times, deep learning has enabled APR, and many tools using the concept of Large Language Models (LLMs) with the Transformer technique have been developed. LLMs are giving better results for many code-related tasks, and researchers have started to use them for APR [12]. Tools such as Codex, CodeBERT, and more recently ChatGPT use LLMs for code fixing.

What makes ChatGPT stand out is its ability to discuss the source code with user interaction. Sobania et al. (2023) [13] conducted an experiment to test the efficiency of ChatGPT in bug fixing compared to other tools like Codex and CoCoNut. About 40 of QuixBugs benchmark problems containing erroneous code were given to ChatGPT to provide solutions. The experiment showed that ChatGPT's performance is similar to other APR tools like Codex. However, when given more information about the problem through its dialogue box, ChatGPT's performance improved, with a success rate of 77.5%.

Although ChatGPT works fine with simple code logic, it can be challenging to describe complex needs, such as designing a web browser where user needs should be satisfied, with the simple, machine-readable instructions that ChatGPT or other AI tools use to produce code [14]. Chatterjee and Dethlefs [15] have claimed that code generated by ChatGPT is also gender and race biased, questioning the efficiency of the model.

Previous works and tools for automated code generation mostly relied on using neural networks [16], [17], which cannot match the current performance of GPT-3 models. Raychev et al. [18] proposed a code completion technique using statistical language models to discover highly rated sentences and recommend code completion suggestions. Sun et al. [19] introduced a novel tree-based neural architecture that incorporates grammar rules and AST structures into the network, and it has been shown to have the best accuracy among all neural network-based code generation methods. Ciniselli et al. [20]

conducted a detailed empirical study on BERT models for code completion and evaluated the percentage of perfect predictions that match developer-written code snippets. However, while BERT models offer a potential solution for code completion, their performance is lower compared to LLM models such as Copilot and ChatGPT.

In summary, there is a scarcity of research that delves into the applications of AI-assisted code generation tools. ChatGPT has risen as a prevalent option among these tools. The current study aims to assess the effectiveness of ChatGPT as a code generation tool. To our knowledge, this is the first research that employs a dataset from social media to evaluate the performance of a code generation tool.

III. METHOD

In this section, we present the proposed data-driven generated coding investigation framework by introducing how to collect data of interest, how to analyze data, and how to interpret findings.

A. Framework Overview

The overview of the proposed framework is illustrated in Figure 2. It consists of three key components: Keyword Selection, Data Collection, and Data Analytics. Unlike traditional user study-based research, the crowdsourcing platform is designed to be flexible and scalable, enabling the study of a large population over a long period of time. We will delve into each component in detail, examining the performance of LLMs in code generation.

B. Keyword Selection for Software Development

To ensure the quality of the collected data, we employ a hybrid approach that combines data-driven keyword expansion and expert-based keyword selection. This approach ensures that the data is comprehensive and precise, eliminating the risk of bias or incompleteness in the selection of query keywords.

As ChatGPT is one of the most popular LLMs that supports code generation, we use *ChatGPT* as the seed keyword to sample Twitter streams, harvesting tweets that mention this term. We then perform topic modeling to determine whether a coding-related topic is present. If a coding-related topic is observed, we add the words belonging to this topic to the expanded keyword set. If a coding-related topic is not observed, we conduct a co-occurrence word analysis and calculate the semantic similarity with the word *coding* to expand the candidate keywords.

However, the data-driven keyword expansion method may result in false positives, i.e., keyword candidates that may not be relevant to AI-based code generation. Therefore, we manually examine all recommended keyword candidates to ensure the quality of the collected data. We first filter out irrelevant keywords and propose multiple combinations of keywords to control the precision of data collection. For example, instead of collecting all postings containing *ChatGPT*, collecting postings containing both *ChatGPT* and *coding* makes the retrieved data more accurate and representative.

Specifically, we leveraged Twitter Streaming APIs to sample tweet streams containing the keyword "ChatGPT" for over 55 hours. In total, we collected 158,452 tweets, including original tweets, retweets, and replies. After removing duplicate tweets, we had 63,716 unique tweets. We then applied the LDA model to infer topics based on these unique tweets, with the hope of discovering programming-related topics. We evaluated the number of topics ranging from 1 to 30 and found that the convergence score achieved a relatively high and stable value with the number of topics set as 22. For more details, please see Figure 10. After examining the 22 topics, we identified one of them as "Programming," consisting of the following words: *ask, stack, knew, write, error, diffus, run, python, stabl, scientist, email, straight, shock, gener, comput, command, use, code, notic, brain, bug, statement, think, dead, question, admit, happen, result, and overflow*.

Combining the words in the topic of *Programming*, we come up with the following keyword list – *algorithm, algorithms, bug, bugs, c#, c++, code, coding, command, commands, compiler, computing, debug, debugging, error, go, interpreter, java, javascript, libraries, php, program, programming, python, r, Ruby, shell, software, sql, stack overflow, swift, test, testing, typescript* – to crawl ChatGPT related code generation posts.

C. Data Collection

Based on the above carefully curated keywords, we leverage two social media platforms, i.e., Twitter and Reddit to collect data for further analytics.

1) Twitter Data: Instead of relying on Twitter Streaming APIs, we opt to use the Twitter Historical Data Search APIs to create our Twitter dataset for the following reasons: 1) The streaming data is time-sensitive, making it impossible to retrieve older data from the debut of ChatGPT if the streaming data collection was not be launched at that time; 2) If we only investigate the latest data (e.g., after Feb 1, 2023), it will introduce bias as we do not know when the performance of ChatGPT's code generation was most widely discussed on social media. On the other hand, the historical tweets span the entire evolution of ChatGPT and provide a sample of user comments since its release, enhancing the representativeness and completeness of the crowdsourced opinions and feedback.

Twitter provides two historical data search APIs, i.e., 30-Day Search API[1] that allows for retrieval of the last 30 days and the Full-archive Search API[2] that provides tweets since 2006 when the first tweet was posted. Since ChatGPT was released on November 30, 2022, we choose the Full-archive Search API to harvest the data. Specially, we use Twitter's Academic Research API, which supports full-archive tweet search, to retrieve ChatGPT-related data from November 30, 2022 to February 1, 2023, with the query configured to only cover English tweets and exclude retweets by setting "-is:retweet lang:en." In addition to the tweet text, we collect related Twitter media information (e.g., posted images) to

---

[1] https://tinyurl.com/2s4xt8r7
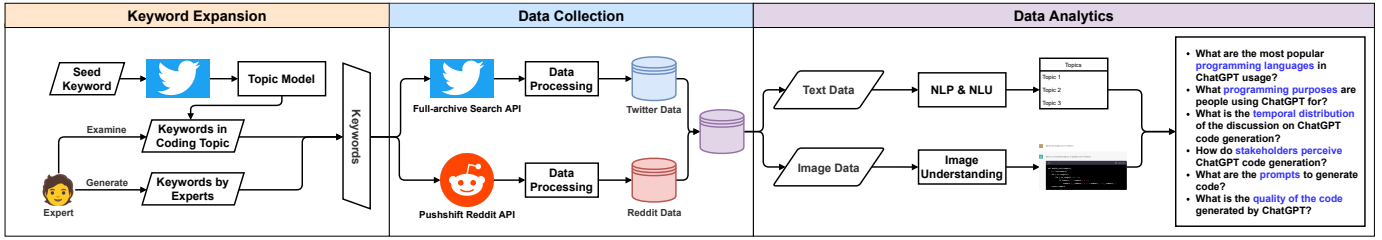[2] https://tinyurl.com/ehbsjx6v

Fig. 2. Overview of the proposed crowdsourcing framework to investigate the programming capabilities of LLMs

support fine-grained analysis. For this study, we collected 316K tweets posted between Dec. 1, 2022, and Jan. 31, 2023.

2) Reddit Data: Unlike Twitter, where the structure is based on users following one another, Reddit is structured around communities where posts on similar topics are grouped together. These communities are referred to as "subreddits" on Reddit. For instance, the subreddit /r/aww is a community where users share cute and cuddly pictures. The initial posts on Reddit are known as "submissions," and the responses to these posts are called "comments."

To assess the performance of ChatGPT in code generation on Reddit, we concentrate on four well-known subreddits, namely /r/ChatGPT, /r/coding, /r/github, and /r/programming. To gather submissions from these subreddits, we use the Search Reddit Submissions Endpoint (/reddit/search/submission) through the Pushshift Reddit API [21]. Similar to Twitter data, multimedia data including images in Reddit submissions are also retrieved. For this study, we collected 3.2K Reddit submissions posted between Dec. 1, 2022, and Jan. 31, 2023, to analyze the code generation performance of ChatGPT.

D. Data Analytics and Pattern Recognition

We primarily employ natural language processing and image understanding techniques to analyze text and image data to uncover insights and identify patterns.

1) Text Based Topic Discovery: To obtain a comprehensive understanding of the use of ChatGPT in code generation on Twitter and Reddit, we employ latent Dirichlet allocation (LDA) [22], a widely used topic modeling technique, to uncover hidden topics in the collected tweets and Reddit submissions. We treat each tweet or submission content as an individual document and the entire collection of tweets or submissions as the corpus of documents. In the text preprocessing stage, we implement commonly used techniques such as removing stop words and frequently occurring words like "ChatGPT," tokenization, and lemmatization of words. We then perform term frequency-inverse document frequency (TF-IDF) on the combined documents to form a TF-IDF-based corpus, on which latent topics are extracted using LDA models. Following previous research on big social data analysis [23], [24], we select the C_v metric to determine the appropriate number of topics. This metric is known to be one of the best coherence measures as it combines normalized pointwise mutual information (NPMI) and cosine similarity [25].

Given that Twitter allows users to utilize #hashtags to indicate related topics and enhance visibility through searches, we also present the distribution of #hashtags in the collected tweets. However, as #hashtags are rarely used on Reddit, we do not perform this analysis for Reddit submissions.

2) Image Understanding: As ChatGPT is a text generative model, it is expected that most images related to ChatGPT, particularly those related to code generation, posted on social media will be text-rich. To make these images more informative and easier to process for downstream tasks, we suggest using an Optical Character Recognition (OCR) based approach to convert the collected images into text. We apply multiple OCR methods, including OpenCV-based pytesseract[3] and deep learning-based easyOCR[4], to the collected image dataset. After thoroughly evaluating the OCR detection results, we choose easyOCR to identify and extract text from the images accurately.

3) Code Reconstruction from Image: To reconstruct the code generated by ChatGPT, it is crucial to identify the images that contain generated code. After examining the screenshots of coding snippets, we found that all ChatGPT-generated code snippets contained the keyword "Copy code" in the top-right corner of the coding block, as shown in Figure 1. Therefore, we selected all images containing the "Copy code" keyword for further analysis.

We proposed two methods to recover the code generated by ChatGPT. The first one is to extract the code directly from the OCR results. We found that it is crucial to address any indentation issues for indentation-sensitive programming languages, such as Python, as a high percentage of errors can occur due to improper indentation. However, automatically indenting any given code can be a complex and challenging task. A simple script that looks for loops and specific statements to increase and decrease the indentation count does not work on all codes, especially if the code has multiple indentation styles and conditional statements.

An alternative method to obtain the code is reproducing it using the identical prompt. Specifically, we can identify the prompt and input it into ChatGPT web services[5] to generate the code. Once we have downloaded the newly produced code, we can assess and evaluate it. In our study, we adopted this reliable method to reconstruct the code generated by ChatGPT.

---

[3]https://pypi.org/project/pytesseract/
[4]https://github.com/JaidedAI/EasyOCR
[5]https://openai.com/blog/chatgpt/

4) Sentiment Analysis: Considering that ChatGPT may trigger diverse emotions in code generation, we do not think the three categories of positive, negative, and neutral can cover all involved emotions. In order to accurately reflect the various and complex emotions expressed in social media users' comments, we choose to categorize them into more inclusive emotions: Happy, Angry, Sad, Surprise, and Fear. To achieve this, we utilize Text2Emotion [26], a Python package capable of analyzing sentiments and categorizing them into the aforementioned five emotions.

5) Code Quality Evaluation: To assess the quality of the code generated by ChatGPT, we are utilizing Flake8 [27], which is a wrapper around PyFlakes, pycodestyle, and Ned Batchelder's McCabe script. Flake8 allows the use of any of these tools by launching Flake8, and it assigns a unique code number to each error code. The output of warnings and errors is displayed per file. We choose Flake8 as our evaluation tool because it is one of the most powerful and flexible tools available, providing a wide range of error codes while remaining fast to run checks. Flake8 is particularly effective when checking for correctness and whitespace issues, making it an ideal choice for our purposes.

IV. EVALUATION AND FINDINGS

In this section, we present the evaluation results and highlight our findings on the performance of code generation by ChatGPT. We summarize the topics discussed in social media posts, the strengths and weaknesses of ChatGPT's code generation capabilities.

A. Programming Language Distribution

ChatGPT supports code generation for multiple programming languages. We illustrate the popularity of the top 12 programming languages across Twitter and Reddit in Figure 3. We can see that python is the most popular language among both communities and far ahead of other languages. Obviously, python has become the top 1 program language in many fields, such as artificial intelligence, machine learning, data analytics, automation, scientific computing, and others. JavaScript, R, and Shell/Bash, among the most popular programming languages nowadays, are also well-supported by ChatGPT.
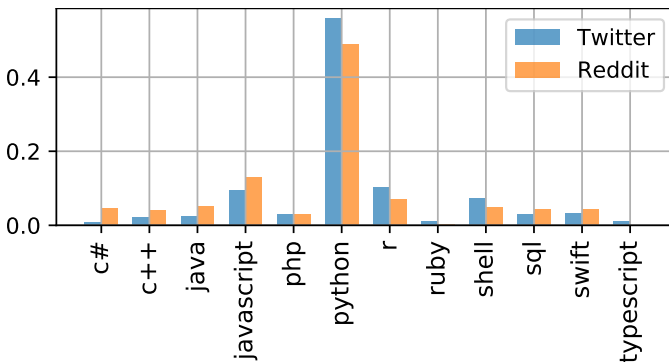
Fig. 3. Programming language distribution

B. Topics Related to Code Generation

We generated topics for the tweets containing keyword "ChatGPT" and programming related words using the LDA model. Based on the coherence score presented in Figure 4, we select 17 topics finally. The 17 topics and the word list of each topic are presented in Table II (see Appendix B). The topic modeling results indicate that ChatGPT has been used for different purposes regarding code generation, such as debugging codes (topic 9, topic 13, topic 17), testing codes/algorithms (topic 5, topic 16), preparing programming interview (topic 2 and topic 4), working on programming-related assignments (topic 3, topic 6), and other related tasks. Twitter users also notice that ChatGPT's capacity in code generation is limited (topic 1). Still, the ethic issues and social responsibility aspect of ChatGPT have not been discussed much among users.
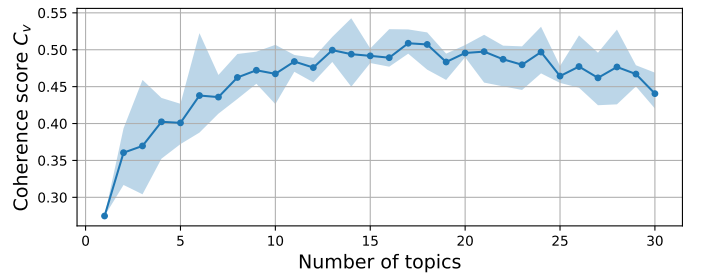
Fig. 4. Coherence scores of LDA with different number of topics

To further investigate the implications and impacts of ChatGPT on different AI technologies, applications, and industries, we extracted the hashtag-based topics, which is shown in Figure 5. The hashtags we use include: #AI, #OPENAI, #Artificialintelligence, #Programming, #Python, #Coding, and others. We group the hashtags into five clusters: ChatGPT, AI & ML & DS, Company, Programming, and Other Tech. Based on the topic frequency in Figure 5, we can see that ChatGPT has a great impact on AI and its related fields. Both academia and IT industry need to pay attention to this new technology.
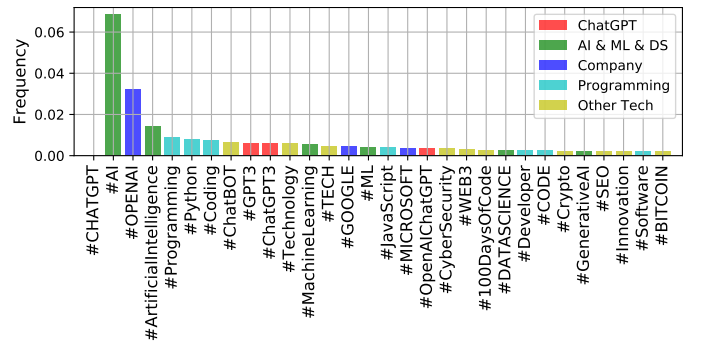
Fig. 5. Hashtag-based topics. We exclude the 35.4% ratio of the #ChatGPT during visualization to prevent it from overpowering other topics

C. Temporal Distribution

Temporal analysis can be used to examine the popularity over time. Figure 6 visualizes the daily distribution of posts on

5

Twitter (blue) and Reddit (yellow) related to ChatGPT's code generation in the first two months after its launch. ChatGPT discussion spread faster on Twitter than Reddit. We observe a peak of the ChatGPT code generation on Twitter and Reddit at the end of the first week of the release of ChatGPT. The popularity decreased from the second week, but somehow still very popular on both platform. Even after two months, the attention on ChatGPT is still stable, indicating ChatGPT is helpful for code generation.
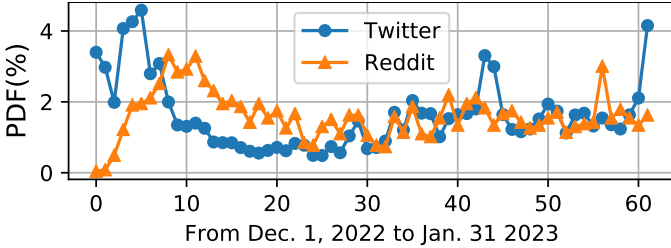
Fig. 6. Daily distribution of posts related to ChatGPT's code generation in the first two months after its launch

D. Sentiment on Code Generation

Figure 7 presents sentiment analysis results for code generation in eight programming languages across two social media platforms: Twitter and Reddit. The emotions were categorized into five distinct groups: happiness, anger, surprise, and fear.

Overall, fear is the most commonly expressed emotion on both platforms for all programming languages, likely due to concerns about ChatGPT's code quality and its potential impact on human jobs. We observed that fear is more frequently expressed for SQL, Java, and C#.

The second most commonly expressed emotion is sadness for Python, JavaScript, R, C++, and Shell on the Twitter platform. This may also be linked to concerns about Chat-GPT's potential impact on human jobs. Surprise is the third most commonly expressed emotion for Python, JavaScript, R, Shell, and C++. The surprise may result from the quality of the generated code. Happiness and anger are the two least frequently observed emotions.

We also compared the sentiment analysis results on both platforms and found similar patterns for all programming languages except SQL and C++. For SQL, Reddit users expressed more sadness than Twitter, possibly due to their greater knowledge about SQL and concerns about the quality of ChatGPT's code generation. Regarding C++, we observed that Reddit users showed more happiness than Twitter users, which may indicate less worry about ChatGPT's potential threat to their jobs.

E. A Public Dataset of Prompts and Generated Code

From the OCR results of Twitter images, we identified and extracted 332 prompts. Figure 8 provides a wordcloud overview of all extracted prompts, where Python-related questions are the most common. In particular, Twitter users prefer words such as "write", "code", "function", and "program" when constructing their coding prompts.

Code	Description	Percentage
E501	line too long (114 >79 characters)	29.39%
E231	missing whitespace after ','	15.54%
E302	expected 2 blank lines, found 1	13.51%
W293	blank line contains whitespace	10.14%
E402	module level import not at top of file	5.41%
E305	expected 2 blank lines after class, found 1	4.73%
E265	block comment should start with '# '	4.39%
W291	trailing whitespace	2.70%
E999	SyntaxError: invalid syntax	2.36%
E227	missing whitespace around bitwise or shift operator	1.69%
W292	no newline at end of file	1.69%
E101	indentation contains mixed spaces and tabs	1.35%
F401	'torch' imported but unused	1.35%
W191	indentation contains tabs	1.35%
W391	blank line at end of file	1.35%
E261	at least two spaces before inline comment	1.01%
E225	missing whitespace around operator	0.68%
F811	redefinition of unused 'pymesh' from line 5	0.34%
E902	TokenError: EOF in multi-line statement	0.34%
F821	undefined name 'output_value'	0.34%
E741	ambiguous variable name 'I'	0.34%

We constructed a dataset of .py files for all Python-related prompts, with each .py file containing the prompt and the corresponding code generated by ChatGPT. Figure 9 shows a sample .py file from the dataset, where the prompt is commented at the beginning of the file. The complete dataset will be publicly released soon to the software engineering community.

F. Code Quality Evaluation

We submitted the Python code snippets generated by Chat-GPT to Flake8 as individual .py files to check for quality and errors. Flake8 identified the error codes for each file, along with the position and description of the error. After evaluating the code snippets using Flake8, we found that the majority of the errors are pycodestyle errors, with code E (80.74%), followed by code W (17.25%). The least number of errors are attributed to pyflake with code F (2.03%). Among the unique error codes, there are 13 for E, with the majority of errors linked to code E501 (line too long). Additionally, there are five unique W codes and three unique F codes. Table IV-F provides a detailed summary of the evaluation results, including the percentage of each Flake8 code for the overall evaluation.

V. CONCLUSION

This paper presents a framework for exploring the code generation capabilities of ChatGPT through the analysis of crowdsourced data on Twitter and Reddit. The results show that Python and JavaScript are the most frequently discussed programming languages on social media and that ChatGPT is used in a variety of code generation domains, e.g., debugging codes, preparing programming interviews, and solving academic assignments. Sentiment analysis reveals that people generally have fears about the code generation capabilities of ChatGPT, rather than feeling happy, angry, surprised, or sad.

Fig. 7. Sentiment analysis results on code generation for eight programming languages

Fig. 8. WordCloud of prompts

Fig. 9. A sample in the public dataset of prompts (Line 3) and generated code (Line 4 - Line 17)

The study also includes the construction of a code generation prompt dataset, which will be made publicly available, and an evaluation of the quality of code generated by ChatGPT using Flake8. We hope this work provides valuable insights into the adoption of ChatGPT in software development and programming education.

APPENDIX

A. Coherence Scores of LDA with Different Number of Topics

One of the most important steps for applying topic modeling such as LDA is to select an appropriate number of topics contained by the corpus [28]. The reason is that choosing too few topics will produce over-broad topics while choosing too many topics will lead to lots of overlapping between topics. In this study, we choose the C_v metric, a widely used coherence measurement to decide the optimal number of topics in our corpus. Topic coherence scores a single topic by combining normalized pointwise mutual information (NPMI) and the cosine similarity between words in the topic [25]. The higher the coherence score, the higher the quality of the generated topics; however, low quality topics may be composed of highly unrelated words that cannot fit into another topic, leading a low coherence score [25]. In our corpus, we evaluated the topic numbers ranging from one to thirty with 500 passes, and we repeated the experiments five times in each step when generating the topics to avoid random errors in C_v metric. Figure 10 presented the evaluation results on all the tweets containing keyword "ChatGPT". In this figure, the horizontal axis indicates the number of topicsm, the vertical axis indicates the coherence score, the top in the shadow represents the max coherence score and the bottom represents the min coherence

score with the number of topics set differently. Since either the selected number of topics (k) is too big (i.e., k > 30) or too small (i.e., k<5) will make the topic interpretation problematic, we finally selected 22 topics for the highest coherence score between 5 to 30 topics.
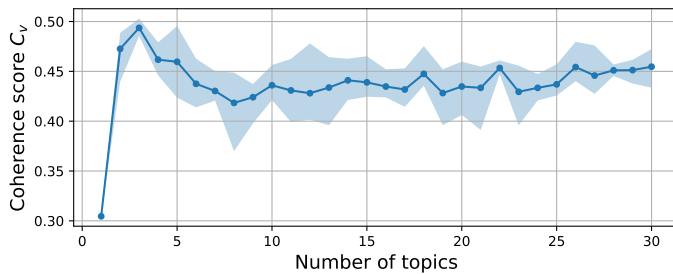

Fig. 10. Coherence scores of LDA with different number of topics

B. LDA Topics Related to Code Generation on Twitter

Table II illustrates the 17 topics inferred by the LDA model from the fine-toned ChatGPT's code generation related tweets. We provided the first 40 words for each topic to demonstrate the most common words. Our analysis shows that ChatGPT has been utilized for various purposes in code generation, including debugging codes (topics 9, 13, and 17), testing codes/algorithms (topics 5 and 16), preparing for programming interviews (topics 2 and 4), working on programming-related assignments (topics 3 and 6), and other related tasks.

REFERENCES

[1] M. E. Kauffman and M. N. Soares, "Ai in legal services: new trends in ai-enabled legal services," *Service Oriented Computing and Applications*, vol. 14, no. 4, pp. 223–226, 2020.

[2] R. Louie, A. Coenen, C. Z. Huang, M. Terry, and C. J. Cai, "Novice-ai music co-creation via ai-steering tools for deep generative models," in *Proceedings of the 2020 CHI conference on human factors in computing systems*, 2020, pp. 1–13.

[3] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," 2022. [Online]. Available: https://arxiv.org/abs/2206.15000

[4] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs experience: Evaluating the usability of code generation tools powered by large language models," in *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3491101.3519665

[5] M. Aljanabi, M. Ghazi, A. H. Ali, S. A. Abed *et al.*, "Chatgpt: Open possibilities," *Iraqi Journal For Computer Science and Mathematics*, vol. 4, no. 1, pp. 62–64, 2023.

[6] L. Avila-Chauvet, D. Mejía, and C. O. Acosta Quiroz, "Chatgpt as a support tool for online behavioral task programming," *Available at SSRN 4329020*, 2023.

[7] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung *et al.*, "A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity," *arXiv preprint arXiv:2302.04023*, 2023.

[8] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[9] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, "Intellicode compose: Code generation using transformer," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1433–1443.

[10] E. Jiang, E. Toh, A. Molina, K. Olson, C. Kayacik, A. Donsbach, C. J. Cai, and M. Terry, "Discovering the syntax and strategies of natural language programming with generative language models," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022, pp. 1–19.

[11] F. F. Xu, B. Vasilescu, and G. Neubig, "In-IDE code generation from natural language: Promise and challenges," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–47, 2022.

[12] C. S. Xia and L. Zhang, "Conversational automated program repair," *arXiv preprint arXiv:2301.13246*, 2023.

[13] D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of chatgpt," *arXiv preprint arXiv:2301.08653*, 2023.

[14] D. Castelvecchi, "Are chatgpt and alphacode going to replace programmers?" *Nature*, 2022.

[15] J. Chatterjee and N. Dethlefs, "This new conversational ai model can be your friend, philosopher, and guide... and even your worst enemy," *Patterns*, vol. 4, no. 1, p. 100676, 2023.

[16] J. Li, Y. Wang, M. R. Lyu, and I. King, "Code completion with neural attention and pointer networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 4159–4165. [Online]. Available: https://doi.org/10.24963/ijcai.2018/578

[17] Z. Sun, Q. Zhu, L. Mou, Y. Xiong, G. Li, and L. Zhang, "A grammar-based structural cnn decoder for code generation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 7055–7062, Jul. 2019. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/4686

[18] V. Raychev, M. Vechev, and E. Yahav, "Code completion with statistical language models," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 419–428. [Online]. Available: https://doi.org/10.1145/2594291.2594321

[19] Z. Sun, Q. Zhu, Y. Xiong, Y. Sun, L. Mou, and L. Zhang, "Treegen: A tree-based transformer architecture for code generation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, pp. 8984–8991, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6430

[20] M. Ciniselli, N. Cooper, L. Pascarella, D. Poshyvanyk, M. D. Penta, and G. Bavota, "An empirical study on the usage of BERT models for code completion," *CoRR*, vol. abs/2103.07115, 2021. [Online]. Available: https://arxiv.org/abs/2103.07115

[21] J. Baumgartner, S. Zannettou, B. Keegan, M. Squire, and J. Blackburn, "The pushshift reddit dataset," in *Proceedings of the international AAAI conference on web and social media*, vol. 14, 2020, pp. 830–839.

[22] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[23] Y. Feng, Z. Lu, Z. Zheng, P. Sun, W. Zhou, R. Huang, and Q. Cao, "Chasing total solar eclipses on twitter: Big social data analytics for once-in-a-lifetime events," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[24] Y. Feng, D. Zhong, P. Sun, W. Zheng, Q. Cao, X. Luo, and Z. Lu, "Micromobility in smart cities: A closer look at shared dockless e-scooters via big social data," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[25] M. Röder, A. Both, and A. Hinneburg, "Exploring the space of topic coherence measures," in *Proceedings of the eighth ACM international conference on Web search and data mining*, 2015, pp. 399–408.

[26] J. Ni, J. Li, and J. McAuley, "Justifying recommendations using distantly-labeled reviews and fine-grained aspects," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 188–197.

[27] T. Ziadé and I. Cordasco, "Flake8: Your tool for style guide enforcement. 2021," *URL: http://flake8. pycqa. org (besucht am 27. 05. 2019)*.

[28] H. Chen, J. Chen, and H. Nguyen, "Demystifying covid-19 publications: institutions, journals, concepts, and topics," *Journal of the Medical Library Association: JMLA*, vol. 109, no. 3, p. 395, 2021.

TABLE II
THE EXTRACTED TOPICS USING THE LDA TOPIC MODEL

Rank	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9	Topic 10	Topic 11	Topic 12	Topic 13	Topic 14	Topic 15	Topic 16	Topic 17
1	capac	softwar	exam	ture	languag	student	haha	nah	code	dey	write	error	que	login	stabl	test	pour
2	bro	engin	school	interview	program	cheat	woke	delet	use	song	code	network	con	simplifi	diffus	trade	de
3	harder	develop	mba	pass	code	teacher	commit	nobodi	ask	profess	use	van	para	rubi	companion	comput	est
4	test	job	test	test	test	essay	broke	everybodi	write	phase	gener	die	una	broken	til	free	que
5	medium	replac	pass	candid	model	assign	annoy	publicii	work	evolut	ask	een	por	battl	ship	money	pa
6	shit	code	law	flutter	use	malici	partner	judg	tri	glad	creat	het	la	prime	plot	pay	sur
7	unlock	googl	intellig	tree	algorithm	school	steroid	rout	help	temp	test	polit	lo	helper	discord	softwar	avec
8	premium	use	pa	newslett	gener	educ	dear	somebodi	program	suffer	python	advent	softwar	flow	jest	paid	le
9	reaction	technolog	artifici	leap	human	use	fli	lambda	time	frequent	program	messag	del	tabl	member	use	une
10	spin	program	professor	conduct	write	malwar	touch	coffe	good	sad	command	occur	assembl	rap	academia	servic	mai
11	eye	tool	wharton	equival	question	detect	outsourc	curios	question	competitor	prompt	doom	como	dog	leak	crypto	qui
12	tab	think	program	holiday	answer	cybersecur	citat	farm	answer	semant	video	met	pero	chain	cite	version	cest
13	sat	take	stock	fail	ask	kid	bisa	revolut	know	pseudo	make	profession	error	odd	skip	bitcoin	par
14	famili	new	univers	extract	data	write	appar	recip	bug	alor	websit	alpha	python	skip	wallet	code	dan
15	tho	search	medic	duck	learn	softwar	greatest	ticket	give	exponenti	app	overload	nut	maker	investig	price	fair
16	write	futur	grade	behav	train	teach	disappoint	nowaday	learn	cryptocurr	content	persist	test	haiku	nie	sell	fait
17	limerick	write	bar	extent	think	colleg	sweet	encount	fix	aux	post	factual	che	framework	ive	cost	test
18	holi	year	busi	siri	softwar	homework	parti	respect	problem	yea	want	yall	robot	conscious	rich	cloud	jai
19	exploit	go	pose	slide	understand	code	untuk	bother	thing	layoff	idea	android	todo	star	laugh	algorithm	plu
20	accept	tech	musk	appl	comput	career	white	straight	gener	ride	tweet	neural	toy	revis	bill	buy	code
21	skynet	stack	educ	ansibl	text	academ	2021	interestingli	make	cave	twitter	dat	inject	workout	workout	employe	vou
22	asset	peopl	softwar	obsess	respons	hacker	sir	imaginari	python	shame	help	subtl	hacer	latex	mom	make	tout
23	alexa	amp	cat	sentient	make	program	pump	hahaha	debug	strang	build	readabl	artifici	extend	elabor	azur	lui
24	lost	make	public	doctor	way	secur	vote	grab	explain	exclus	script	apolog	dia	analys	unreal	power	comm
25	weekend	test	licens	studio	need	plagiar	yang	irrelev	error	season	tri	men	pued	five	visit	open	bir
26	ask	product	quantum	entiti	natur	test	anybodi	peer	googl	ca	work	infinit	dune	watermark	graduat	compani	son
27	drive	overflow	countri	hunt	see	univers	older	certif	even	inner	see	pleas	sobr	boom	staff	need	mon
28	refresh	way	invest	roll	peopl	attack	threw	cook	test	theori	check	scam	algo	monitor	strength	program	sui
29	aspect	world	startup	examin	creat	new	third	superior	find	tou	new	center	outlook	observ	cycl	amazon	bug
30	singular	gener	stream	stress	good	hack	screw	salari	need	pipelin	text	trump	crawl	associ	item	user	peut
31	comprehens	work	tesla	nation	one	gener	footbal	admin	solv	trop	articl	crawl	crucial	brother	king	token	bien
32	casual	see	score	januari	tool	threat	as	merg	better	donner	tool	whoever	per	coolest	own	million	quil
33	travel	chang	firm	sec	googl	puzzl	crime	club	day	ecosystem	let	kan	monkey	number	coach	sourc	quand
34	server	time	lab	rubber	new	creat	clone	showcas	one	streamlin	blog	friendli	esta	santiago	gym	way	san
35	war	help	till	alongsid	convers	crack	bright	dream	realli	nick	bot	contact	muy	press	ace	write	gen
36	fuck	learn	learn	premier	machin	ban	woman	counter	exampl	weather	copi	code	tien	sandbox	ale	posit	non
37	steal	skill	lawyer	matur	base	email	manner	cup	copilot	evil	thread	ook	ser	linear	delight	invest	bon
38	weird	smart	final	satisfi	amp	develop	gui	sleep	want	meanwhil	imag	net	substanti	resist	god	meta	cett
39	ile	contract	world	declar	design	concern	numer	winner	actual	mond	give	mental	powerpoint	screenshot	sparrow	go	python
40	verbatim	compani	founder	wisdom	know		worst	upcom	way	aris	chat	insist	hay	atm	boy	billion	ture

9