mathematics

MDPI

Article

SCC-GPT: Source Code Classification Based on Generative Pre-Trained Transformers

Mohammad D. Alahmadi [1,*], Moayad Alshangiti [1] and Jumana Alsubhi [2]

[1] Department of Software Engineering, College of Computer Science and Engineering, University of Jeddah, Jeddah 23890, Saudi Arabia; mshangiti@uj.edu.sa
[2] School of Computing, University of Georgia, Athens, GA 30602, USA; jha40847@uga.edu
* Correspondence: mdalahmadi@uj.edu.sa

Abstract: Developers often rely on online resources, such as Stack Overflow (SO), to seek assistance for programming tasks. To facilitate effective search and resource discovery, manual tagging of questions and posts with the appropriate programming language is essential. However, accurate tagging is not consistently achieved, leading to the need for the automated classification of code snippets into the correct programming language as a tag. In this study, we introduce a novel approach to automated classification of code snippets from Stack Overflow (SO) posts into programming languages using generative pre-trained transformers (GPT). Our method, which does not require additional training on labeled data or dependency on pre-existing labels, classifies 224,107 code snippets into 19 programming languages. We employ the `text-davinci-003` model of ChatGPT-3.5 and postprocess its responses to accurately identify the programming language. Our empirical evaluation demonstrates that our GPT-based model (SCC-GPT) significantly outperforms existing methods, achieving a median F1-score improvement that ranges from +6% to +31%. These findings underscore the effectiveness of SCC-GPT in enhancing code snippet classification, offering a cost-effective and efficient solution for developers who rely on SO for programming assistance.

Keywords: SCC (Source Code Classification); NLP (Natural Language Processing); Large Lagnuage Model (LLM)

MSC: 68T01

Citation: Alahmadi, M.D.; Alshangiti, M.; Alsubhi, J. SCC-GPT: Source Code Classification Based on Generative Pre-Trained Transformers. *Mathematics* **2024**, *12*, 2128. https://doi.org/10.3390/math12132128

Academic Editor: Shaomin Wu

Received: 19 May 2024
Revised: 28 June 2024
Accepted: 5 July 2024
Published: 7 July 2024

1. Introduction

Developers frequently rely on online resources to seek assistance and guidance for their programming tasks, using online sources such as Stack Overflow (SO) or video programming tutorials. When searching for solutions or information, developers often include specific programming languages in their queries to narrow down the search results and find relevant resources. Platforms like Stack Overflow (SO) provide tagging functionality to help with indexing the immense amount of information available. Developers can manually tag their questions or posts with the appropriate programming language of the attached code snippet to make them more discoverable and to increase the chances of receiving relevant answers [1,2]. However, despite the availability of tagging features and its importance, not all questions or posts are adequately tagged with the proper programming language, and some may even be tagged incorrectly [3,4].

To address this issue, it is crucial to automate the classification of source code snippets in Stack Overflow (SO) posts into the correct programming language. Extensive research efforts have been dedicated to utilizing machine learning (ML) and Natural Language Processing (NLP) techniques for accurately classifying source code into appropriate programming languages [5–8]. However, it is important to note that much of the prior work in this area has primarily focused on classifying large source code files obtained from datasets like GitHub, which inherently possess more features than the code snippets

typically found in SO posts [9]. Consequently, applying these existing methods directly to code snippets may yield suboptimal results due to the variations in code length and contextual information. Those code snippets could also be found in programming video tutorials where classifying the code could help in mining the programming videos such as in extracting source code [10–12], classifying video frames [13–15], and finding relevant software artifacts for video contents [16–18].

Several studies have focused on the classification of code snippets in Stack Overflow (SO) posts into 21 different programming languages [4,9,19]. In their research, Alrashedy et al. [9,19] gathered a substantial dataset, comprising 232,727 SO posts that contained code snippets and were already tagged with 21 programming languages. They proposed the Source Code Classification (SCC) model [19], utilizing the Multinomial Naive Bayes (MNP) [20] algorithm to train their dataset. In a subsequent study, they introduced SCC++ [9], employing gradient boosting algorithms such as XGBoost [21] and Random Forest (RF) [22] to enhance the learning process. Another study by Yang et al. proposed DeepSCC [4], which was designed to further improve the performance of predicting the programming language of the same dataset by fine-tuning the RoBERTa model [23]. Despite achieving reasonable results, the practicality and feasibility of these models may be limited for two reasons: (i) they were primarily trained on a dataset with potentially inaccurate labels, as they were manually tagged by the creators of the SO posts, and (ii) developers cannot directly utilize these models for assistance in tagging and/or classifying code snippets. Consequently, addressing these limitations and exploring alternative approaches becomes crucial for improving the practicality and usability of code snippet classification.

Recently, large-scale language models, particularly generative pre-trained transformers (GPT) [24], have emerged as powerful tools for tackling real-world challenges. These models have undergone extensive training on vast amounts of data, enabling them to develop a comprehensive understanding of language patterns and structures. One key advantage of these models is their remarkable ability to generalize across diverse tasks without requiring extensive fine-tuning specific to each task. This versatility has generated considerable interest among researchers, leading to their application in various domains, including code generation [25], code and model completion [26–28], and bug fixing [29,30]. Moreover, Large Language-based Models like ChatGPT have demonstrated their potential in assisting developers without the need to train models from scratch or rely on manually labeled datasets. However, the effectiveness of these models in classifying code snippets remains an underexplored area, presenting an opportunity for further investigation and potential advancements in the Source Code Classification (SCC) domain.

In this paper, we propose a novel and cost-effective approach that harnesses the power of generative pre-trained transformers (GPTs) for the automatic classification of code snippets obtained from Stack Overflow (SO) into 19 distinct programming languages. Our study focuses on evaluating and analyzing the performance of ChatGPT-3.5, utilizing the advanced `text-davinci-003` model, to accurately predict the programming language of a substantial dataset comprising 224,107 code snippets. We utilized ChatGPT by requesting it to identify the programming language used in each code snippet. We then postprocessed the model's response by removing any unrelated text and keeping only the detected programming language.

We conducted a comprehensive performance comparison against well-established traditional ML-based models, including SCC [19], SCC++ [9], a state-of-the-art DL-based model called DeepSCC [4], and the Programming Languages Identification (PLI) [31] tool, renowned for its exploitation of diverse statistical and syntactic features. Our extensive evaluation revealed that the Source Code Classification using a GPT-based model, referred to as SCC-GPT, significantly outperformed all existing approaches on the 19 programming languages. Notably, the median F1-score of SCC-GPT exhibited remarkable improvements, ranging between +6% and +31%, compared to the aforementioned models.

We have included in our online appendix https://zenodo.org/records/11832019 (accessed on 4 July 2024) (i) the entire dataset that contains code snippets and the cor-

responding programming language and (ii) the script with the prompt we formulated, the postprocessing script, and the results analysis.

The remainder of the paper is organized as follows: Section 2 reviews related work. Section 3 outlines dataset extraction and processing, along with machine learning classifiers. Section 4 presents our results. Finally, threats to validity and conclusions are outlined in Sections 5 and 6.

2. Related Work

2.1. Source Code Classification

Prior investigations into source code classification have taken various routes. However, most of them have not fully leveraged the potential of modern, transformer-based language models in classifying smaller code snippets. For example, Khasnabish et al. [5] assembled over 20,000 source code files from multiple GitHub repositories. Employing a Bayesian classifier, their model aimed to predict 10 programming languages. In another approach, Gilda [6] proposed a neural network-based model for source code classification. Van Dam and Zaytsev [7] explored software language identification using natural language classifiers. Their experiment further highlighted the potential of leveraging language processing techniques. In addition, the programming language identification tool by Algorithmia [31] offers a practical solution for language detection. Nevertheless, these models' efficacy in the context of smaller code snippets, like those typically found on Stack Overflow, hasn't been comprehensively evaluated. Moreover, it is crucial to emphasize that the labeling of data must be carried out prior to any analysis or utilization.

On the other hand, Baquero et al. [8] focused on extracting knowledge from Stack Overflow posts to predict the programming language. Their method, while innovative, lacked the ability to handle isolated code snippets without contextual clues being provided in Stack Overflow posts. A couple of studies by Alrashedy et al. [9,19] demonstrated the application of machine learning in predicting programming languages from Stack Overflow snippets and questions. However, their machine learning models required comprehensive training and lacked robustness across different languages. Lastly, Yang et al. [4] developed DeepSCC, a classification approach based on a fine-tuned RoBERTa model, and demonstrated its capabilities with the aid of natural language statistical models, although using a fine-tuned RoBERTa model requires extra network training. Moreover, it is crucial to emphasize that the labeling of data must be carried out prior to any analysis or utilization used in the aforementioned work.

In contrast to prior studies, our approach employs a Large Language Model for source code classification, surpassing state-of-the-art methods in efficiency and accuracy without the need for additional training. While existing studies predominantly concentrated on classifying large source code files from GitHub, our research uniquely addresses the diverse and fragmented nature of code snippets on platforms like Stack Overflow.

2.2. ChatGPT in Software Engineering

The use of Large Language Models (LLMs) like ChatGPT in software engineering has seen a surge of interest in recent years. Xu et al. [25] carried out a systematic evaluation of Large Language Models for code. Their findings showed that such models could provide valuable insights when applied to code analysis and development. The potential of few-shot learning was explored by Chaaben et al. [28], aiming to automate model completion. They demonstrated how prompt learning could make substantial contributions to model completion tasks, highlighting the power of LLMs in learning from limited examples. In another study, Kang et al. [29] described how LLMs could serve as few-shot testers. They explored the potential of LLMs in bug reproduction, opening new horizons for automated testing in software engineering. Sobania et al. [32] analyzed the performance of ChatGPT in automatic bug fixing. Their research emphasized the model's capabilities in identifying and correcting software errors, hinting at the future of automated debugging. Akli et al. [33] presented a unique application of few-shot learning to predict flaky test

categories. Their innovative approach leveraged the capacity of LLMs to learn from sparse examples, improving the prediction of unstable tests. Lyu et al. [34] introduced Chronos, a time-aware zero-shot identification tool for libraries from vulnerability reports. Their work further expands the application scope of LLMs in vulnerability detection and remediation. Le and Zhang [35] explored log parsing with prompt-based few-shot learning. This approach demonstrates the adaptability of LLMs in processing log files, a critical aspect of system monitoring and diagnostics. Nashid et al. [36] investigated retrieval-based prompt selection for code-related few-shot learning. Their work elucidates the importance of careful prompt selection in few-shot learning tasks. Finally, Siddiq et al. [37] studied zero-shot prompting for code complexity prediction using GitHub Copilot. Their results showcased the potential of LLMs in understanding and predicting the complexity of coding tasks.

Each of these studies underline the growing importance of LLMs in various areas of software engineering, from code analysis and testing to vulnerability detection and code complexity prediction. Their collective findings suggest the burgeoning potential of LLMs to revolutionize many aspects of software engineering practices. Thus, building upon the successful applications of LLMs in various aspects of software engineering, our research seeks to leverage the power of Large Language Models for the task of source code classification. By tapping into the ability of these models to understand and generate human-like text, we aim to bring a new perspective and potentially improved performance to the task of classifying code snippets from a wide array of programming languages.

3. Empirical Study

In this section we present the dataset and our methodology for assessing ChatGPT's SCC.

3.1. Dataset Description

The dataset was collected by Alrashedy et al. [9,19] from Stack Overflow, a popular online platform where developers post their programming-related questions and answers. As of July 2017, Stack Overflow had over 37 million posts, with a significant number of these tagged with different programming languages. The data dump from July 2017 was used and questions were selected so that they only had one programming language tag and at least one code snippet with more than two lines of code. The aim was to avoid potential confounding factors.

The data dump from July 2017 was used and the questions were selected so that they only had one programming language tag and at least one code snippet with more than two lines of code. The aim was to avoid potential confounding factors. There were 12,000 questions per language extracted, but, for Markdown and Lua, there were only 1210 and 8107 questions, respectively.

In this paper, similar to a previous study [4], we exclusively used code snippets from 19 programming languages. More specifically, we omitted code snippets associated with (i) HTML (12,000 code snippets, $\sim$5%), as our manual observation revealed that many of these snippets also included segments of CSS and JavaScript, and (ii) Markdown, which is significantly lower than that of other languages (only 1210 code snippets, $\sim$0.5%).

3.2. SCC-GPT Evaluation Methodology

During the evaluation of SCC-GPT, we leveraged the SCC [4] benchmark set, comprising 224,107 pairs of code snippets and their corresponding language types. Our evaluation process unfolded in three key steps. Initially, we formulated queries that prompted ChatGPT to classify a given code snippet into one of the 19 programming languages: Bash, C, C#, C++, CSS, Haskell, Java, Javascript, Lua, Objective-C, Perl, PHP, Python, R, Ruby, Scala, SQL, Swift, and VB.net. Subsequently, we submitted each code snippet to ChatGPT-3.5 utilizing the text-davinci-003 backend model. Finally, we postprocessed the ChatGPT replies, retaining only the predicted programming language, and saved it for

subsequent comparison with the ground-truth programming language. This meticulous approach ensures a comprehensive evaluation of SCC-GPT's proficiency in programming language classification.

4. Results and Discussion

4.1. Evaluation Metrics

To assess the effectiveness of a classifier in categorizing source code snippets from 19 programming languages, we employed precision, recall, and F1-score as evaluation metrics. These evaluations are determined by considering True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) values. By reporting the F1-score, we can summarize the classifier's performance with a single metric that balances precision and recall.

Precision measures the accuracy of positive predictions made by the classifier. It calculates the ratio of true positives (TP) to the sum of true positives and false positives (FP). A low precision means that a classifier is returning a large number of false positives. Precision is defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

Recall measures the ability of the classifier to correctly identify positive instances. It calculates the ratio of true positives (TP) to the sum of true positives and false negatives (FN). A high recall value indicates that the classifier has a low rate of false negatives, meaning that it can effectively identify most positive instances. Recall is defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

F1-score provides a balanced evaluation of the classifier's performance, considering both precision and recall simultaneously. In particular, we used the F1-score or balanced F-score, which is the harmonic mean of precision and recall, defined as follows:

$$\mathrm{F1} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

4.2. SCC-GPT Experimental Results

Table 1 presents a comparison of the accuracy results in terms of F1-score between SCC-GPT and previous approaches for source code snippet classification. The table includes performance metrics for various programming languages and algorithms, including SCC++ XGBoost [9], SCC++ RF [9], SCC MNP [19], PLI [31], DeepSCC [4], and SCC-GPT.

The analysis of the experimental findings reveals that SCC-GPT outperforms the baseline algorithms and achieves the highest performance in source code classification. Across the programming languages examined, SCC-GPT consistently demonstrates superior accuracy compared to the other algorithms. Notably, it achieves substantial improvements in the F1-score metric, indicating its effectiveness in accurately classifying code snippets. SCC-GPT achieves a maximum performance improvement of 5% in terms of F1-score compared to the baseline algorithms. For specific languages, SCC-GPT achieves outstanding performance. In particular, it surpasses the other algorithms in classifying Bash, C, C#, C++, Haskell, Java, JavaScript, Lua, Objective-C, Perl, Python, R, Ruby, Scala, SQL, Swift, and VB.net code snippets, as evidenced by its higher F1-score. These results highlight SCC-GPT's superior ability to capture and learn the deep semantics of code, leading to more accurate classification outcomes.

However, the performance of SCC-GPT in CSS and PHP can be attributed to the challenges posed by the intertwined nature of these languages with others. The presence of CSS within HTML or JavaScript code makes it challenging to accurately distinguish CSS-specific snippets. Similarly, PHP's integration with other scripting languages adds

complexity to the classification process, as it becomes more difficult to differentiate PHP code from code written in alternative scripting languages. This ambiguity in distinguishing CSS from JavaScript or HTML and PHP from other scripting languages contributes to the performance variations observed in SCC-GPT's classification results for CSS and PHP.

Comparing the performance of SCC-GPT with the other baselines in the field of source code classification, it becomes apparent that SCC-GPT consistently outperforms them. This indicates the effectiveness of the SCC-GPT approach, which leverages advanced techniques such as generative pre-trained transformers to capture deep code semantics, resulting in improved classification accuracy.

In summary, the experimental results clearly demonstrate the superiority of SCC-GPT over the baseline algorithms in source code snippet classification. SCC-GPT's exceptional performance across various programming languages showcases its effectiveness in accurately categorizing code snippets. The findings highlight the potential of the two-way transformer encoder approach in capturing the intricate semantics of code, leading to enhanced classification outcomes.

Table 1. Comparing the accuracy (F1-score) results of SCC-GPT with previous work for source code snippet classification.

Lang./Algos	SCC++ XGBoost [9]	SCC++ RF [9]	SCC MNP [19]	PLI [31]	DeepSCC [4]	SCC-GPT
Bash	0.80	0.85	0.76	0.67	0.87	0.91
C	0.76	0.81	0.76	0.56	0.81	0.87
C#	0.80	0.78	0.79	0.51	0.83	0.89
C++	0.49	0.73	0.51	0.65	0.82	0.90
CSS	0.87	0.77	0.86	0.30	0.87	0.70
Haskell	0.90	0.78	0.89	0.67	0.93	0.94
Java	0.70	0.76	0.70	0.46	0.86	0.92
Javascript	0.80	0.74	0.78	0.48	0.82	0.89
Lua	0.86	0.70	0.84	0.50	0.91	0.94
Objective-C	0.65	0.88	0.57	0.77	0.92	0.95
Perl	0.77	0.41	0.74	0.69	0.86	0.91
PHP	0.75	0.88	0.74	0.62	0.83	0.88
Python	0.89	0.79	0.88	0.69	0.86	0.92
R	0.78	0.78	0.77	0.72	0.92	0.93
Ruby	0.72	0.72	0.70	0.43	0.88	0.91
Scala	0.78	0.81	0.76	0.72	0.93	0.93
SQL	0.67	0.79	0.65	0.50	0.85	0.93
Swift	0.87	0.89	0.84	0.54	0.94	0.95
VB.net	0.86	0.77	0.83	0.60	0.88	0.88
Average	0.77	0.77	0.76	0.58	0.87	0.90

Figure 1 visually represents the performance distribution of various source code classification models, including SCC++ XGBoost [9], SCC++ RF [9], SCC MNP [19], PLI [31], and DeepSCC [4]. The boxplot highlights key statistical metrics for each model, such as minimum, maximum, median, and average F1-score values across the 19 programming languages. This graphical representation offers a comprehensive overview of the comparative effectiveness of the models, providing insights into their variability and overall performance in code snippet classification. Specifically, as observed in Figure 1, SCC-GPT, our proposed approach, demonstrates notable strengths across all key metrics. With a minimum F1-score of 0.70, SCC-GPT surpasses the minimum scores of several comparative models, emphasizing its robustness in handling diverse programming languages. The maximum F1-score of 0.95 attests to the exceptional performance of SCC-GPT, outperforming other models in capturing the complexities of code snippets. The median and average F1-score values of 0.91 and 0.90, respectively, further underscore the consistency and effectiveness of SCC-GPT in programming language classification. These results collectively affirm the superior performance and reliability of our proposed SCC-GPT approach in comparison to established models in the field.

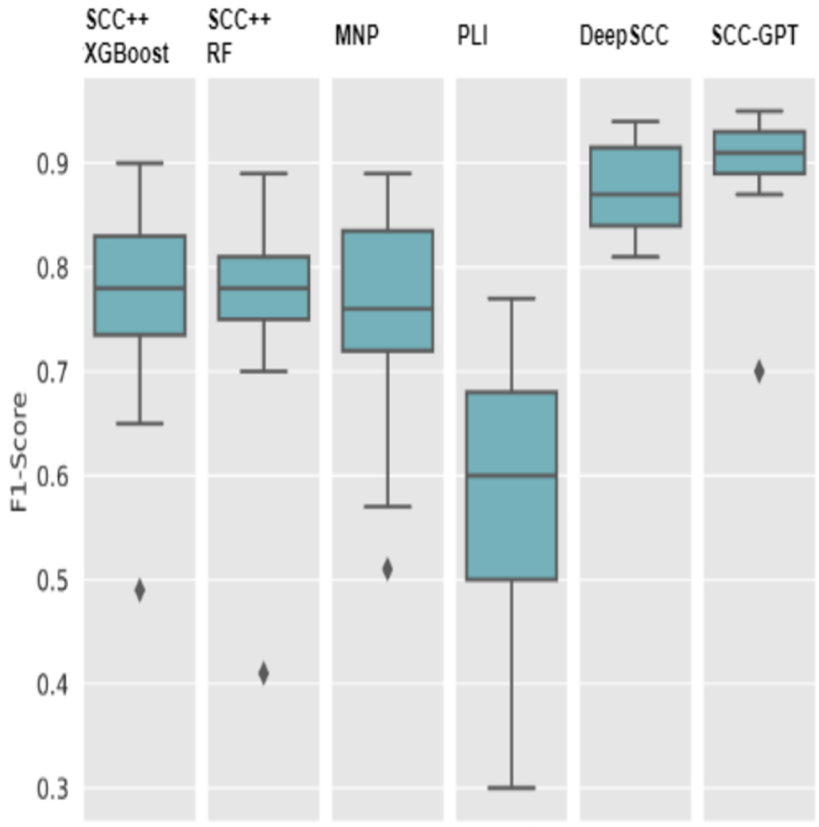

Figure 1. Boxplots of the F1-score results in predicting the 19 programming languages used in our study (∼224k code snippets) using SCC-GPT compared to five baselines: SCC++ XGBoost [9], SCC++ RF [9], SCC MNP [19], PLI [31], and DeepSCC [4].

4.3. A Discussion of ChatGPT Answers

In this section, we present three illustrative examples (shown in Figure 2) featuring queries accompanied by code snippets, each followed by the corresponding responses generated by ChatGPT for the classification of programming languages. The purpose is to showcase instances where ChatGPT provides incorrect or misleading language classifications, offering insights into its performance nuances. These examples serve as valuable illustrations to understand potential challenges and areas for improvement in the model's proficiency as a programming language classifier.

In Figure 2a, the code snippet, written in VB.NET, employs server-side code blocks within the ASP.NET Razor view engine. Despite the distinctive VB.NET syntax, ChatGPT incorrectly classifies the code as belonging to the ASP.NET Razor view engine used in conjunction with the C# programming language. This misclassification underscores the model's challenge in precisely identifying VB.NET code constructs and its tendency to associate ASP.NET Razor syntax more strongly with C# rather than VB.NET.

In Figure 2b, the code snippet presents an HTML structure, defining a container with an id of 'bottomcontent,' styled using inline CSS properties. The container is configured for a fixed position at the bottom-left corner of its containing element. Within this container, there is a child div named 'bottomleftbutton' containing the text "stuff here" and set with a left margin. However, ChatGPT's classification as "CSS" in the response is incorrect, as the provided code is, in fact, an HTML structure incorporating inline CSS styling. This misclassification highlights the challenge in distinguishing between HTML and CSS, especially when examining code snippets that involve both languages.

In Figure 2c, ChatGPT has made an incorrect classification by identifying it as Swift, while the actual programming language is Objective-C. This misclassification can be attributed to the syntactic similarities between Objective-C and Swift, leading to confusion

in certain scenarios. Specifically, Swift allows for interoperability with Objective-C code, and their syntax can overlap, making it challenging to distinguish between the two languages accurately. In this case, the use of square brackets and the NSString class may have contributed to the misclassification. This underscores the importance of considering contextual cues and nuances in the code when relying on syntax-based language classification, especially in cases where languages share similar syntax features.

> **Prompot:** Specify the programming language (just mention the name) for the following code snippet given that it should belong to one of the 19 following programming languages (Bash, C, C#, C++, CSS, Haskell, Java, Javascript, Lua, Objective-C, Perl, PHP, Python, R, Ruby, Scala, SQL, Swift, VB.net).

```
Code Snippet:
<% Dim FormObject As Form = (Form)Model %>

<%  Dim FormObject As Form = DirectCast(Model, Form)%>

Response:
this code snippet belongs to the asp.net razor view engine, which
    is used in conjunction with the c# programming language
```

(a) An incorrect ChatGPT classification for a VB.net code snippet

```
Code Snippet:
<div id='bottomcontent' style="float: left; position: absolute;
    left: 0; bottom: 0; background-color:#0b0b0b; height:100px;
    width:796px;">

<div id="bottomleftbutton" style="margin-left: 25px;">stuff here
    </div>
</div>

Response:
CSS
```

(b) An incorrect ChatGPT classification for an HTML code snippet

```
Code Snippet:
static NSString const *const viewControllerClassName[3] = {@"one"
    , @"two", @"three"};
const NSString *className = viewControllerClassName[1];
Class class = NSClassFromString((NSString *)className);
UIViewController *controller = [[class alloc]
    initWithContainer:self];

Response:
this code snippet belongs to Swift
```

(c) An incorrect ChatGPT classification for a Swift code snippet

Figure 2. Three examples of queries, each accompanied by code snippets, and the incorrect corresponding responses processed with ChatGPT for classifying programming languages.

To investigate the robustness of the SCC-GPT model to noisy or incomplete code snippets, we conducted a detailed analysis of the correlation between the length of code (measured in the number of words) and the correctness of predictions across various programming languages. We calculated the Pearson correlation coefficient and p-value

for each language, enabling us to determine the strength and statistical significance of these relationships.

Our results revealed that, for most programming languages, including Bash, C++, C, Haskell, JavaScript, Java, Lua, Objective-C, Perl, PHP, Python, Ruby, R, Scala, SQL, Swift, and VB.NET, there is a statistically significant positive correlation between the number of words in the code snippet and the correctness of the model's predictions. This suggests that, as the length of the code increases, the likelihood of correct predictions by SCC-GPT also increases. Conversely, for C# and CSS, we observed a significant negative correlation, indicating that longer code snippets are associated with a decrease in prediction accuracy.

The findings of our study have significant implications for software engineering practices, particularly in automated code analysis, bug detection, and code complexity prediction. The superior performance of SCC-GPT in accurately classifying code snippets enhances the reliability of automated tools in these areas. For automated code analysis, accurate classification ensures that tools can correctly identify and analyze code from different programming languages, improving the precision of code reviews and analysis. In bug detection, the model's ability to accurately identify the programming language can enhance the performance of bug-finding algorithms by applying language-specific heuristics and patterns. For code complexity prediction, the improved classification accuracy supports better identification of complex code segments, enabling more targeted and effective code optimization and refactoring efforts. Overall, SCC-GPT's ability to leverage generative pre-trained transformers for precise code classification contributes to more efficient and reliable software engineering workflows.

5. Threats to Validity

We acknowledge potential threats to the validity of our study. Firstly, ChatGPT is a dynamically evolving model, and, during our research, a significant update was released. While we found similar classification results before and after the update, future releases could yield different outcomes. Moreover, as ChatGPT allows for interactive conversations, the choice of questions asked may influence the results. To mitigate this, we reported the questions that we asked to ensure that there would not be any significant impact on the outcomes.

Furthermore, ChatGPT sometimes outputs more than the expected programming language, which may be considered incorrect if we are checking for an exact match. To overcome this issue, we instructed ChatGPT more explicitly to only output the programming language name, and complemented this approach by implementing a postprocessing stage where the model's responses are analyzed to only include the exact match of the programming language.

Additionally, the results obtained might be influenced by factors such as the programming language. Some programming languages, like CSS and PHP, have specific characteristics that can impact the classification results. CSS, often embedded within HTML, presents challenges in accurately differentiating CSS-specific snippets from HTML or JavaScript code. Similarly, PHP's integration with other scripting languages adds complexity to the classification process, making it more difficult to differentiate PHP code from code written in alternative scripting languages.

ChatGPT may struggle with code snippets that contain multiple programming languages due to context mixing, where different languages (e.g., HTML with embedded JavaScript) are blended, making accurate classification challenging. Limited training data on mixed-language snippets further reduces the model's ability to generalize, while syntax similarities between languages can lead to misclassifications. To address these issues, incorporating specialized preprocessing steps to separate different languages within a snippet, enhancing training datasets with more examples of mixed-language snippets, and employing ensemble models that combine predictions from language-specific classifiers can improve accuracy.

Finally, the size and distribution of the dataset can impact the model's performance and generalizability. In our study, the dataset comprised 224,107 code snippets across 19 programming languages, with some languages having fewer examples. This imbalance can lead to biased learning, where the model performs well on languages with more data and poorly on those with less, increasing the risk of overfitting for languages with fewer examples. To mitigate these issues, we took steps to ensure that the data are as well-balanced as possible and maintained an acceptable minimum number of examples for each programming language by excluding Markdown language, which included only 1210 code snippets compared to 12,000 code snippets for the other programming languages. Additionally, we leveraged GPT's pre-training on diverse data for better generalization and used the F1-score to ensure fair performance evaluation across languages.

6. Conclusions and Future Work

In this study, we addressed the critical challenge of accurately classifying code snippets from Stack Overflow posts into their respective programming languages. Acknowledging the limitations of manual tagging and recognizing the variations in code length and context, we introduced a novel approach that utilizes generative pre-trained transformers (GPT). Leveraging the capabilities of ChatGPT-3.5 with the `text-davinci-003` model, our proposed model, SCC-GPT, demonstrated remarkable effectiveness in classifying 224,107 code snippets across 19 programming languages.

The empirical evaluation revealed significant improvements in classification accuracy, with SCC-GPT consistently outperforming existing methods. The median F1-score improvement, ranging from +6% to +31%, underscores the practicality and efficiency of our GPT-based model. This study not only advances the state-of-the-art in code snippet classification but also addresses the unique challenges posed by varied snippet lengths and contextual intricacies in online programming forums.

As developers increasingly rely on platforms like Stack Overflow for code-related queries, our findings contribute a valuable tool in enhancing the accuracy of programming language classification. The success of SCC-GPT opens avenues for improved resource discovery and effective knowledge sharing within the programming community, marking a substantial step toward more precise and automated code snippet tagging.

The future work directions for the SCC-GPT model are promising and offer numerous opportunities for extending its application. One potential direction is using SCC-GPT for large-scale software repository mining and analysis. As the first step for many mining and analysis tasks is to identify the programming language, accurately doing so can help provide insights into programming language trends, code reuse patterns, and developer practices, thereby informing better decision-making for project management and tooling. Another direction is integrating SCC-GPT with Integrated Development Environments (IDEs) to offer real-time language classification and contextual code suggestions, improving coding efficiency and reducing errors. This integration can also facilitate automated refactoring and code optimization by accurately identifying language-specific patterns and best practices. Additionally, GPT's application can be extended to suggest tags beyond programming languages based on the textual information in Stack Overflow posts. This expansion may leverage GPT's capabilities to provide valuable insights and enhance the tagging process in diverse contexts, further broadening the utility of the model.

Author Contributions: Conceptualization, M.D.A.; Methodology, M.D.A.; Validation, M.A.; Formal analysis, M.A.; Data curation, J.A.; Writing—review & editing, M.A.; Visualization, J.A.; Funding acquisition, M.D.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the University of Jeddah, Jeddah, Saudi Arabia, under grant No. (UJ-23-RSP-2). The authors, therefore, thank the University of Jeddah for its technical and financial support.

Data Availability Statement: The data presented in this study are openly available in (https://zenodo.org/records/11832019).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Stanley, C.; Byrne, M.D. Predicting tags for stackoverflow posts. In Proceedings of the ICCM, Ottawa, ON, Canada, 11–14 July 2013; Volume 2013.
2. Beyer, S.; Pinzger, M. Synonym suggestion for tags on stack overflow. In Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, Florence, Italy, 18–19 May 2015; pp. 94–103.
3. Ye, D.; Xing, Z.; Li, J.; Kapre, N. Software-specific part-of-speech tagging: An experimental study on stack overflow. In Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, 4–8 April 2016; pp. 1378–1385.
4. Yang, G.; Zhou, Y.; Yu, C.; Chen, X. DeepSCC: Source Code Classification Based on Fine-Tuned RoBERTa. *arXiv* **2021**, arXiv:2110.00914.
5. Khasnabish, J.N.; Sodhi, M.; Deshmukh, J.; Srinivasaraghavan, G. Detecting programming language from source code using Bayesian learning techniques. In Proceedings of the Machine Learning and Data Mining in Pattern Recognition: 10th International Conference, MLDM 2014, St. Petersburg, Russia, 21–24 July 2014; Proceedings 10; pp. 513–522.
6. Gilda, S. Source code classification using Neural Networks. In Proceedings of the 2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE), Nakhon Si Thammarat, Thailand, 12–14 July 2017; pp. 1–6.
7. Van Dam, J.K.; Zaytsev, V. Software language identification with natural language classifiers. In Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Osaka, Japan, 14–18 March 2016; Volume 1, pp. 624–628.
8. Baquero, J.F.; Camargo, J.E.; Restrepo-Calle, F.; Aponte, J.H.; González, F.A. Predicting the programming language: Extracting knowledge from stack overflow posts. In Proceedings of the Advances in Computing: 12th Colombian Conference, CCC 2017, Cali, Colombia, 19–22 September 2017; Proceedings 12; pp. 199–210.
9. Alrashedy, K.; Dharmaretnam, D.; German, D.M.; Srinivasan, V.; Gulliver, T.A. Scc++: Predicting the programming language of questions and snippets of stack overflow. *J. Syst. Softw.* **2020**, *162*, 110505. [CrossRef]
10. Alahmadi, M.D.; Alshangiti, M. Optimizing OCR Performance for Programming Videos: The Role of Image Super-Resolution and Large Language Models. *Mathematics* **2024**, *12*, 1036. [CrossRef]
11. Alahmadi, M.D. VID2XML: Automatic Extraction of a Complete XML Data From Mobile Programming Screencasts. *IEEE Trans. Softw. Eng.* **2022**, *49*, 1726–1740. [CrossRef]
12. Alahmadi, M.D. VID2META: Complementing Android Programming Screencasts with Code Elements and GUIs. *Mathematics* **2022**, *10*, 3175. [CrossRef]
13. Ott, J.; Atchison, A.; Harnack, P.; Bergh, A.; Linstead, E. A deep learning approach to identifying source code in images and video. In Proceedings of the 15th International Conference on Mining Software Repositories, Gothenburg, Sweden, 28–29 May 2018; pp. 376–386.
14. Ott, J.; Atchison, A.; Harnack, P.; Best, N.; Anderson, H.; Firmani, C.; Linstead, E. Learning lexical features of programming languages from imagery using convolutional neural networks. In Proceedings of the 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), Gothenburg, Sweden, 27 May–3 June 2018.
15. Alahmadi, M.; Hassel, J.; Parajuli, B.; Haiduc, S.; Kumar, P. Accurately predicting the location of code fragments in programming video tutorials using deep learning. In Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering—PROMISE'18, Oulu, Finland, 10 October 2018; pp. 2–11.
16. Ellmann, M.; Oeser, A.; Fucci, D.; Maalej, W. Find, understand, and extend development screencasts on YouTube. In Proceedings of the 3rd ACM SIGSOFT International Workshop on Software Analytics, Paderborn, Germany, 4 September 2017; pp. 1–7.
17. Nong, C.; Zhang, Q.; Huang, L.; Cui, D.; Zheng, Q.; Liu, T. FVT: A fragmented video tutor for" dubbing" software development tutorials. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), Montréal, QC, Canada, 25–31 May 2019; pp. 95–99.
18. Ponzanelli, L.; Bavota, G.; Mocci, A.; Di Penta, M.; Oliveto, R.; Hasan, M.; Russo, B.; Haiduc, S.; Lanza, M. Too long; didn't watch!: Extracting relevant fragments from software development video tutorials. In Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), Austin, TX, USA, 14–22 May 2016; pp. 261–272. [CrossRef]
19. Alreshedy, K.; Dharmaretnam, D.; German, D.M.; Srinivasan, V.; Gulliver, T.A. [Engineering Paper] SCC: Automatic Classification of Code Snippets. In Proceedings of the 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM), Madrid, Spain, 23–24 September 2018; pp. 203–208. [CrossRef]
20. Rish, I. An empirical study of the naive Bayes classifier. In Proceedings of the IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence, Seattle, WA, USA, 4–10 August 2001; Volume 3, pp. 41–46.
21. Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794.
22. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
23. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv* **2019**, arXiv:1907.11692.
24. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.

25. Xu, F.F.; Alon, U.; Neubig, G.; Hellendoorn, V.J. A systematic evaluation of large language models of code. In Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, San Diego, CA, USA, 13 June 2022; pp. 1–10.
26. Vaithilingam, P.; Zhang, T.; Glassman, E.L. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In Proceedings of the Chi Conference on Human Factors in Computing Systems Extended Abstracts, San Jose, CA, USA, 28 April–3 May 2022; pp. 1–7.
27. Dakhel, A.M.; Majdinasab, V.; Nikanjam, A.; Khomh, F.; Desmarais, M.C.; Jiang, Z.M.J. Github copilot ai pair programmer: Asset or liability? *J. Syst. Softw.* **2023**, *203*, 111734. [CrossRef]
28. Chaaben, M.B.; Burgueño, L.; Sahraoui, H. Towards using Few-Shot Prompt Learning for Automating Model Completion. In Proceedings of the ICSE-NIER '23: Proceedings of the 45th International Conference on Software Engineering: New Ideas and Emerging Results, Melbourne, Australia, 17–19 May 2022.
29. Kang, S.; Yoon, J.; Yoo, S. Large Language Models are Few-shot Testers: Exploring LLM-Based General Bug Reproduction, In Proceedings of the IEEE/ACM International Conference on Software Engineering 2023 (ICSE 2023), Melbourne, Australia, 14–20 May 2023.
30. Prenner, J.A.; Babii, H.; Robbes, R. Can OpenAI's codex fix bugs? an evaluation on QuixBugs. In Proceedings of the Third International Workshop on Automated Program Repair, Pittsburgh, PA, USA, 19 May 2022; pp. 69–75.
31. Programming Language Identification Tool. 2023. Available online: https://www.algorithmia.com/ (accessed on 4 February 2023).
32. Sobania, D.; Briesch, M.; Hanna, C.; Petke, J. An Analysis of the Automatic Bug Fixing Performance of ChatGPT. Available online: https://arxiv.org/abs/2301.08653 (accessed on 5 May 2024).
33. Akli, A.; Haben, G.; Habchi, S.; Papadakis, M.; Traon, Y.L. Predicting Flaky Tests Categories using Few-Shot Learning. In Proceedings of the 2023 IEEE/ACM International Conference on Automation of Software Test (AST), Melbourne, Australia, 15–16 May 2022.
34. Lyu, Y.; Le-Cong, T.; Kang, H.J.; Widyasari, R.; Zhao, Z.; Le, X.B.D.; Li, M.; Lo, D. Chronos: Time-aware zero-shot identification of libraries from vulnerability reports. In Proceedings of the ICSE' 23: Proceedings of the 45th International Conference on Software Engineering, Melbourne, Australia, 14–20 May 2023.
35. Le, V.H.; Zhang, H. Log Parsing with Prompt-based Few-shot Learning. In Proceedings of the ICSE' 23: Proceedings of the 45th International Conference on Software Engineering, Melbourne, Australia, 14–20 May 2023.
36. Nashid, N.; Sintaha, M.; Mesbah, A. Retrieval-based prompt selection for code-related few-shot learning. In Proceedings of the 45th International Conference on Software Engineering (ICSE'23), Melbourne, Australia, 14–20 May 2023.
37. Siddiq, M.L.; Samee, A.; Azgor, S.R.; Haider, M.A.; Sawraz, S.I.; Santos, J.C. Zero-shot Prompting for Code Complexity Prediction Using GitHub Copilot. In Proceedings of the 2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE), Melbourne, Australia, 20–20 May 2023.