

Assessing the Code Clone Detection Capability of Large Language Models

Zixian Zhang*

School of Computer Science, University of Galway
Ireland
z.zhang15@universityofgalway.ie

Takfarinas Saber†

School of Computer Science, University of Galway
Ireland
takfarinas.saber@universityofgalway.ie

Abstract

This study aims to assess the performance of two advanced Large Language Models (LLMs), GPT-3.5 and GPT-4, in the task of code clone detection. The evaluation involves testing the models on a variety of code pairs of different clone types and levels of similarity, sourced from two datasets: BigCloneBench (human-made) and GPTCloneBench (LLM-generated). Findings from the study indicate that GPT-4 consistently surpasses GPT-3.5 across all clone types. A correlation was observed between the GPTs' accuracy at identifying code clones and code similarity, with both GPT models exhibiting low effectiveness in detecting the most complex Type-4 code clones. Additionally, GPT models demonstrate a higher performance identifying code clones in LLM-generated code compared to humans-generated code. However, they do not reach impressive accuracy. These results emphasize the imperative for ongoing enhancements in LLM capabilities, particularly in the recognition of code clones and in mitigating their predisposition towards self-generated code clones—which is likely to become an issue as software engineers are more numerous to leverage LLM-enabled code generation and code refactoring tools.

Keywords: Code Clone Detection, Large Language Models (LLMs), GPT-3.5, GPT-4, Semantic Analysis.

1 Introduction

The detection of code clones as well as the assessment of code similarity hold considerable importance in the field of software engineering. Identifying and managing code clones is crucial for maintaining code quality and integrity [10]. It aids in reducing redundancy, preventing bugs, and ensuring consistency across the codebase.

*Also affiliated with CRT-AI, the Science Foundation Ireland Centre for Research Training in Artificial Intelligence.

†Also affiliated with Lero, the Science Foundation Ireland Research Centre for Software.

Furthermore, the assessment of code similarity helps detect code leaks/plagiarism [4, 14] and improve automatic code generation [32–34]. Extensive research efforts have been dedicated to code clone detection methodologies over the decades. Despite these efforts, identifying semantic clones—code pairs characterized by low textual similarity—remains a big challenge.

In the wake of artificial intelligence (AI) revolution, scholars have sought solutions through AI-based methods. Increasingly, Machine Learning (ML) and Deep Learning (DL) techniques have been harnessed to unearth code snippets bearing semantic or syntactic similarities, utilizing tools such as Convolutional Neural networks (CNNs) [21, 30], Recurrent Neural Networks (RNNs) [40], Graph Neural Networks (GNNs) [8, 22], and transformers [41]. Nonetheless, there have been no studies that specifically explore the application of Large Language Models (LLMs) in detecting code clones.

Recent years have witnessed a rapid advancement in the field of Large Language Models (LLMs), characterized by their increasing complexity and capabilities. This evolution led to a wide range of new applications in various domains including healthcare [15, 23], machine translation [17, 42], and recommendation systems [9, 36]. In comparison to other domains addressed by LLMs, software engineering presents a unique set of challenges. The precision required in coding, coupled with the necessity for logical consistency, error-free execution, performance, quality, maintainability, and evolution, makes software engineering a significantly more complex domain for LLMs.

Several studies have investigated the performance of LLMs in some aspects of software engineering. These studies have endeavored to explore and evaluate the efficacy of LLMs in understanding and generating code [11, 24, 35], generation of code documentation [20] or repairing code [19]. To the best of our knowledge, the study by Wang et al. [37] is the only to evaluate the

performance of LLMs on code clone detection. However, this study is limited to ChatGPT, and does not explore diverse types of code clones or multiple sources of datasets (including real-world code clones and clones generated by the GPT model itself).

To fill this gap, in this paper, we aim to systematically evaluate the capabilities of two LLM models (i.e., GPT-3.5 and GPT-4) in identifying code clones. Furthermore, as Integrated Development Environments (IDEs) become integrated with LLM tools (e.g., GPT-4 and Microsoft Copilot) and software engineers are more numerous to leverage their capabilities for various software engineering tasks (e.g., code generation and refactoring), we would like to assess whether there is a difference between the performance of LLM models at identifying human-generated code clones in comparison to LLM-generated code clones.

Our paper aims at answering the following Research Questions (RQs):

- **RQ1:** How does the performance of GPT-3.5 compare to the performance of GPT-4 across various code clone types and code similarity levels?
- **RQ2:** Do GPT models exhibit different performances when assessing human-generated versus LLM-generated code clones? If so, how do these differences manifest between GPT-3.5 and GPT-4?

The rest of the paper is organized as follows. Section 2 summarises the background of our study. Section 3 describes the approaches we used to select data and design GPT prompt. In Section 4, we analyze the performance of GPT through the data we selected. Finally, Section 5 concludes the paper.

2 Background

In this section, we describe the background of our study in two parts: Code Clones and Generative Pretrained Transformers.

2.1 Code Clones

A sequence of source code is known as a code fragment that is identical or very similar to another code fragment, this pair is known as clone pairs. These pairs are similar in terms of functionality, structure, or both. These pairs can be found within the same file, across different files in a single project, or even across projects. There are various clone types based on clone pair similarity. We use the definition of clones which has been widely accepted by many scholars [28]:

- **Type-1 (T1):** syntactically identical code segments, with the exception of variations in whitespace, layout, and comments [5].
- **Type-2 (T2):** syntactically identical code segments, with the sole distinctions being in identifier names and literal values, alongside the variances characteristic of Type-1 clones, such as those in whitespace, layout, and comments [5].
- **Type-3 (T3):** syntactically similar code segments, exhibiting differences at the statement level. These segments demonstrate variations wherein statements are added, modified, and/or removed in relation to each other, in addition to the distinctions present in both Type-1 and Type-2 clones, such as disparities in whitespace, layout, comments, identifier names, and literal values [27].
- **Type-4 (T4):** while syntactically dissimilar, code segments implement identical functionality [27].

Basically, the definition of Type-1 clone and Type-2 clone, known as syntactic code clones, are pretty clear, grounded in the textual similarity of code fragments. This phenomenon is commonly observed in practices involving copying and pasting. Conversely, the classification of Type-3 and Type-4 clones, also referred to as semantic code clones, is always controversial. Distinct from syntactic clones, which are identified based on textual similarity, semantic clones may exhibit similar functionality but are often implemented using diverse syntactic structures.

In this study, we adopt the categorization framework of BigCloneBench (BCB) for code clones, which is predicated on their percentage of similarity. Clones are classified as Very-Strongly Type-3 (VST3) when they exhibit a similarity ranging from 90% (inclusive) to 100%. Clones falling within the 70-90% similarity bracket are categorized as Strongly Type-3 (ST3), those within 50-70% as Moderately Type-3 (MT3), and clones with a similarity percentage between 0-50% are designated as Weakly Type-3 or Type-4 (WT3/4).

2.2 Generative Pretrained Transformers (GPTs)

With the success of Transformer-based LLMs, the field of natural language processing (NLP) has witnessed remarkable achievements. Brown et al. [6] introduce the GPT-3.5 model, this model, boasting 175 billion parameters, demonstrated the potential of scaling up parameters in auto-regressive language models for improved performance across a variety of NLP tasks.

Following this, GPT-3.5 emerged as a refined iteration of its predecessor. Developed by Ouyang et al.[25], GPT-3.5 incorporated reinforcement learning from human feedback, enhancing its capabilities and efficiency.

GPT-4, as introduced by OpenAI [1] in 2023, assumed the mantle as the successor to GPT-3.5. Although its exact parameter count was not disclosed, it is presumed to be larger and more sophisticated than its predecessors. GPT-4's evaluation encompassed a diverse array of many NLP datasets, covering tasks from Question Answering (QA) and Natural Language Inference (NLI) to Reading Comprehension (RC). Uniquely, GPT-4 was also tested on sets of exams originally designed for humans, showing its advanced understanding and response capabilities.

3 Approach

In this section we describe our proposed approach to evaluate the performance of LLM models at identifying code clones in two parts: Code Clone Dataset, and GPT Prompt Engineering.

3.1 Datasets

In our study, we use two particular datasets: (i) BigCloneBench is the most used in the literature and combines human-engineered code clones, and (ii) a new dataset that combines GPT-engineered code clones.

3.1.1 BigCloneBench [31]. A widely-used clone detection benchmark in code clone detection tasks. It is a comprehensive collection of 8 million validated clones within IJaDataset-2.0 [16], a repository containing 25,000 open-source Java systems. This benchmark covers both intra-project and inter-project clones across four primary clone types, spanning the entire range of clone syntactical similarity.

3.1.2 GPTCloneBench [3]. A comprehensive benchmark designed to evaluate semantic and cross-language code clones using GPT-3.5 [6] and SemanticCloneBench [2]. GPTCloneBench leverages GPT-3.5's capabilities to generate semantic and cross-language clones from code fragments in SemanticCloneBench. GPTCloneBench includes several true semantic clone pairs, false semantic pairs, and cross-language clones across four programming languages (Java, C, C#, and Python).

3.1.3 Data Selection. To maximize the efficiency of evaluating GPT models' performance in code clone detection while minimizing experimental costs (specifically, the usage of API tokens), we have implemented a

Table 1. Clone Pair Summary

Dataset/Clone Type	T1	T2	VST3	ST3	MT3	WT3/T4
BCB	300	300	300	300	300	300
GPTCloneBench	/	/	/	/	300	300

meticulous clone selection process before their analysis with GPT models. Within the constraints of our budget, we chose 300 samples for input into the GPT models for each dataset and clone type. For the BCB dataset, which comprises multiple open-source Java repositories and contains numerous replicated code clones, our initial step involves filtering out these replicated clones to ensure the uniqueness of the code clones submitted to the GPT models. Subsequently, for each clone type, we endeavor to maintain representative code pairs with diverse similarities across the 300 code pairs, aiming for a balanced representation of clone variations.

In the case of GPTCloneBench, which serves as a benchmark for comparing the performance of GPT models between LLM-generated and human-made code clones, we employ the same selection process as with BCB. Following this, for each clone type within GPTCloneBench, we choose 300 samples to align the code size distribution for each clone type group as closely as possible with the code size distribution of the corresponding clone type in the selected samples from the BCB dataset.

The details of the selected data are presented in Table 1. Owing to the scarcity of clone types T1, T2, VST3, and ST3 in the GPTCloneBench dataset, our selection was limited to 300 samples each for MT3 and WT3/T4 types from this dataset. Additionally, it is important to note that both datasets are devoid of negative samples, which are non-clone pairs. Consequently, this study solely encompasses positive samples, necessitating the use of True Positive, False Negative, and Recall as the primary evaluation metrics.

3.1.4 Selected Data Profile. The distribution of code similarity for clone types WT3/T4, MT3, ST3, and VST3 in the selected data from BCB is illustrated in Figure 1. This figure reveals that the data selected from BCB for these types contains diverse similarities, signifying that the selected data is apt for reflecting GPT models' performance across a diverse range of code pair similarities.

The Program size distributions for the data selected from BCB and GPTCloneBench are depicted in Figure 2.

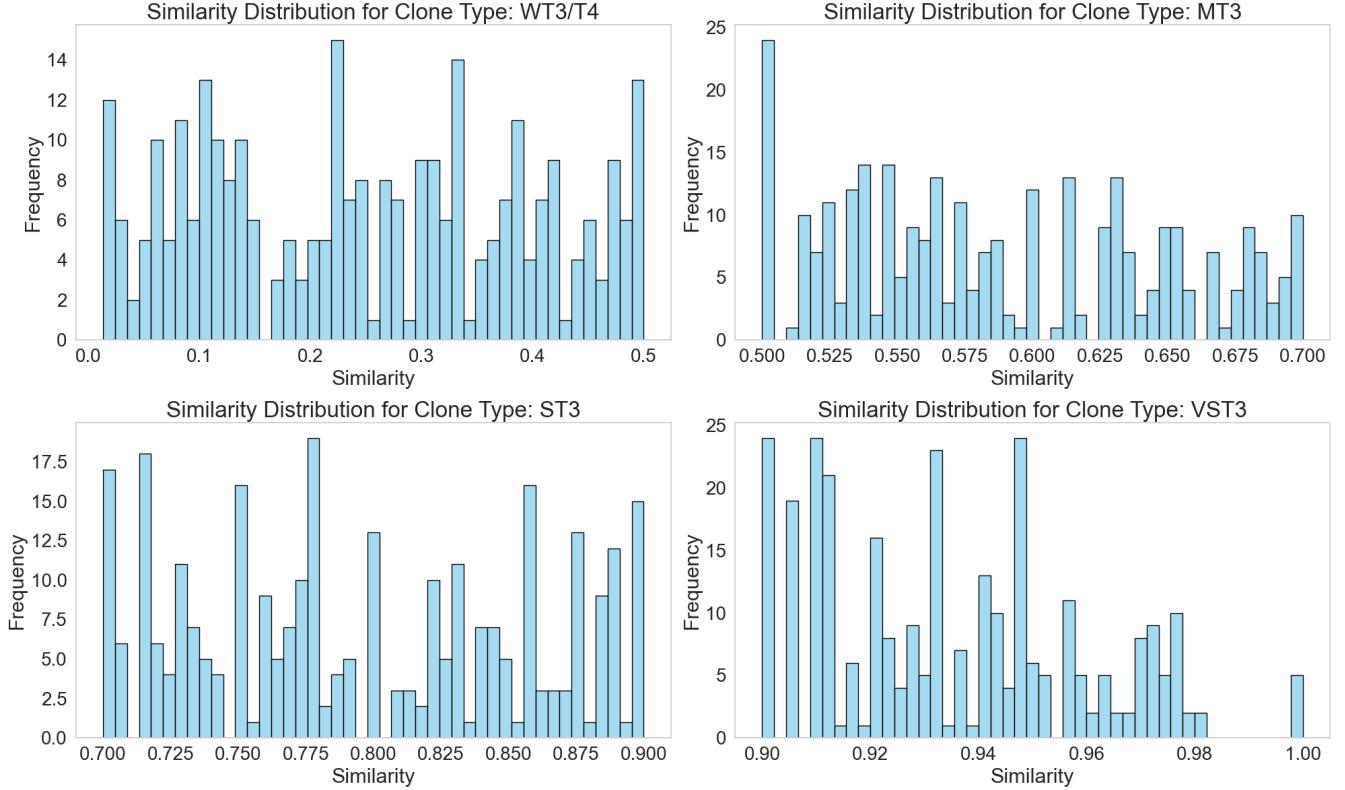


Figure 1. BCB Similarity Distribution per Clone Type

The bulk of the codes fall within the size ranges 400-600 and 250-750 Bytes for BCB and GPTCloneBench respectively.

3.2 GPT Prompt Engineering

A clear and accurate prompt is essential in the execution of natural language processing tasks. A prompt constitutes a set of input instructions or guidelines aimed at eliciting a specific output or accomplishing a particular task. It may take the form of a question, a statement, or an instruction. Prompts play a pivotal role in facilitating interaction with LLMs, thereby aiding in achieving the intended outcome from the model's response.

There exist various techniques to define prompts, (e.g., zero-shot [26, 39], one-shot [13], few-shot [12], chain of thought [38], self-improving [7], or analogy-based [18]). Zero-shot employs direct instructions without the need for pre-training data or prior knowledge input. One-shot enables a model to perform tasks based on a single example. Few-shot trains a model using only a handful of examples for new tasks. Chain of thought prompts encourage models to detail their reasoning step by step. Self-improving prompts guide models to

critique and enhance their outputs over time. Analogy-based prompts help models solve problems by drawing comparisons to familiar scenarios. Various prompt techniques are suitable for different situations. The selection of a prompt can substantially influence the performance of the model.

3.2.1 Impractical One-Shot Prompt. We first analyzed the feasibility of a one-shot prompt using this prompt: “*Determine whether two provided code snippets are code clones. Output yes or no with no explanation.*”.

We conducted a manual evaluation of the one-shot prompt technique using several code pair samples. Our findings indicate that the GPT models do not yield consistent results from this one-shot prompt approach, showing whether the model provides explanations for its judgment or exhibits misunderstandings regarding code clones.

3.2.2 Selecting Few-Shot Prompt. We shifted our approach to a few-shot prompt technique to accurately assess and contrast the performance of the GPT models. Specifically, we constructed our prompt with simple instructions, an input sample, and definitions of code

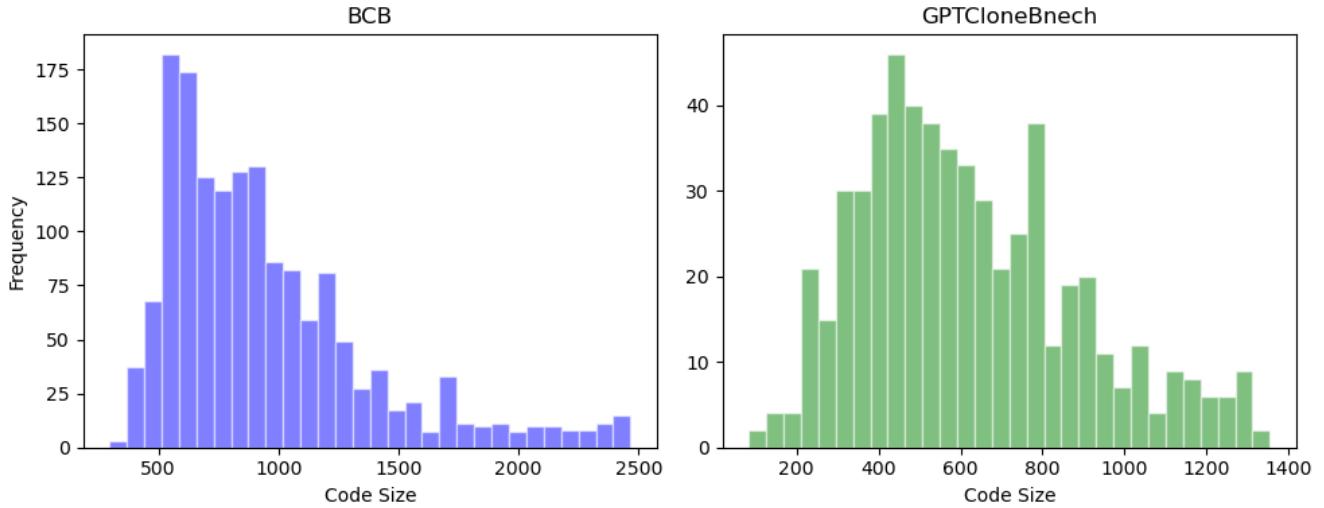


Figure 2. BCB and GPTCloneBench Code Size Distributions

clones to direct GPT models in determining whether a given code pair is a clone or not. Figure 3 shows an example of the creation of the prompt with a code pair. The prompt is segmented into three distinct parts. The first section comprises the main instruction, which outlines the basic task description and provides a definition of code clones to the GPT model. The second section offers an example, supplying GPT with a sample input and its corresponding output. The final section poses a query to the GPT model, featuring two code snippets labeled with numbers and colons, requesting the GPT to render a determination on whether the specified code pair is a clone. By employing multiple code pair samples, we find that this few-shot prompt yields stable and expected results from the GPT model.

4 Results

In this section, we answer the two research questions defined in Section 1 based on our experimental analysis.

4.1 LLM Performance at Detecting Clone Clones

In our initial comparison of code clone detection performance between GPT-3.5 and GPT-4, we focused on the distributions of true positive and false negative answers across four different Clone Types, utilizing data selected from BCB. The findings are presented in Table 2. For clone Type T1, both GPT-3.5 and GPT-4 achieved a 100% rate of true positives with zero false negatives, demonstrating their proficiency in recognizing Type-1 clones. In the case of clone Type T2, GPT-3.5 exhibited a lower recall of 0.56 in contrast to GPT-4, which

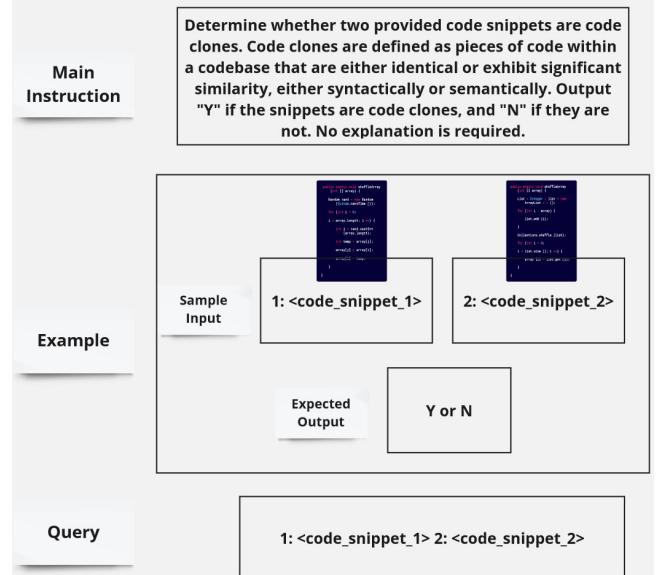


Figure 3. Few-Shot GPT Prompt

achieved 0.86, indicating a notably higher success rate for GPT-4 with this clone type.

For the more semantically complex clone types, VST3, ST3, and MT3, the performance of the GPT-3.5 model declined progressively, with recall generally falling below 0.50. Conversely, the GPT-4 model displayed significantly superior performance, with all recalls exceeding 0.85. For clone Type-4, characterized by pair similarities of less than 50%, both models showed relatively low performance levels; however, GPT-4 still outperformed GPT-3.5 with a recall of 0.23.

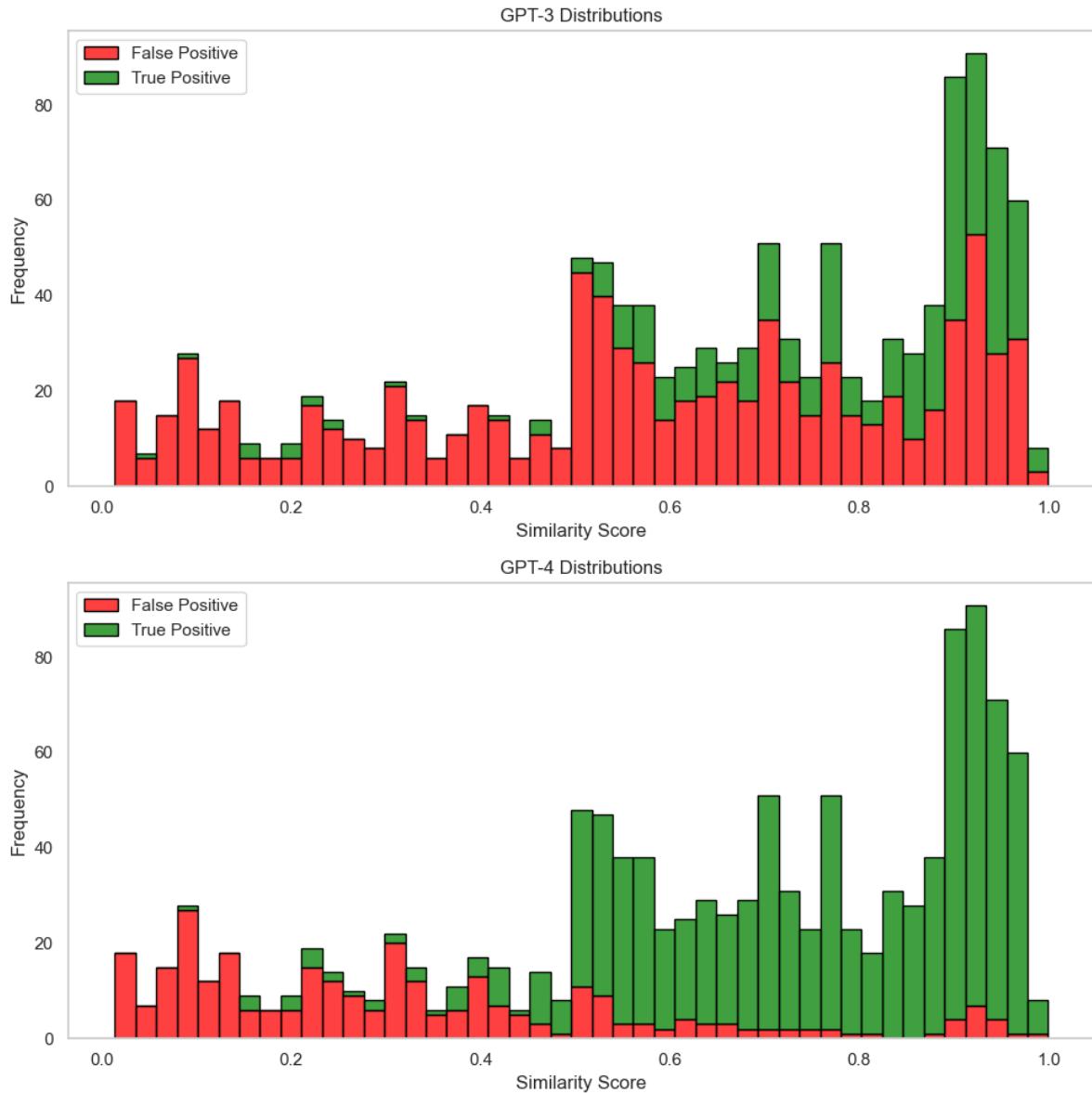


Figure 4. Performance of LLM models per Code Pair Similarity

Our analysis further extends to comparing GPT-3.5 and GPT-4 in terms of their performance relative to the distribution of code pair similarity, as detailed in Figure 4. We notice a pronounced increase in the frequency of true positives within the higher similarity score range (notably from 0.5 to 1.0), corresponding to clone Type-3. This observation suggests a more substantial accumulation of true positives within this range for GPT-4 compared to GPT-3.5, hinting at GPT-4's enhanced capability in accurately identifying code clones with relatively higher similarity.

In the segment pertaining to clone Type-4, where the similarity score trends towards 0.5, the data indicates a predominance of false positives over true positives for both models. Nonetheless, there is an observable decline in the frequency of positive identifications as the similarity score diminishes towards 0.5. Despite this, GPT-4's performance remains superior to that of GPT-3.5 in this specific region, underscoring its improved effectiveness in distinguishing code clones, even within lower similarity scores.

Table 2. Performance of LLM Models on Different Clone Types

Clone Type	Model	TP	FP	Recall
T1	GPT-3.5	300	0	1.00
	GPT-4	300	0	1.00
T2	GPT-3.5	169	131	0.56
	GPT-4	259	41	0.86
VST3	GPT-3.5	156	144	0.52
	GPT-4	283	17	0.94
ST3	GPT-3.5	133	167	0.44
	GPT-4	290	10	0.97
MT3	GPT-3.5	70	230	0.23
	GPT-4	262	38	0.87
WT3/T4	GPT-3.5	20	280	0.07
	GPT-4	68	232	0.23

Table 3. Performance of LLM models on BCB (Human-Generated) and GPTCloneBench (LLM-Generated)

Clone Type	Model	Data Source	TP	FP	Recall
MT3	GPT-3.5	BCB	70	230	0.23
		GPTCloneBench	207	93	0.69
	GPT-4	BCB	262	38	0.87
		GPTCloneBench	252	48	0.84
WT3/T4	GPT-3.5	BCB	20	280	0.07
		GPTCloneBench	174	126	0.58
	GPT-4	BCB	68	232	0.28
		GPTCloneBench	232	68	0.77

Answer to RQ1: The study finds that GPT-4 consistently outperforms GPT-3.5 across all clone types, particularly for code pairs exhibiting low similarity, highlighting GPT-4's advanced ability to understand semantic nuances within the code.

4.2 Difference Between Human-Made and LLM-Generated Code Clones

Table 3 presents the comparative analysis results from different data sources across various clone types (MT3 and WT3/T4) and GPT models (GPT-3.5 and GPT-4). The data sources in question include BCB, which consists of human-generated clones, and GPTCloneBench, comprising LLM-generated clones. The results demonstrate that the LLM-generated GPTCloneBench yields superior performance in terms of higher recall rates compared to the human-generated BCB data across both clone types. This suggests that GPT models exhibit enhanced proficiency in recognizing code clones

that they themselves have generated, as opposed to real-world code clones. Nonetheless, for GPT-4, the difference in performance between BCB and GPTCloneBench is less marked in recognizing MT3 code clones, indicating its robustness in clone detection across sources.

Despite the disparities in evaluation data, we have documented the performance metrics of learning-based code clone detection algorithms for comparison. For reference, existing code clone detection techniques achieve recall rates of 0.94 and 0.45 for clone Type-4 on the BCB [41] and GPTCloneBench [29] datasets, respectively. These references provide further evidence of a bias within GPT models towards more accurately identifying code clones that they themselves have generated, as opposed to recognizing clones originating from real-world scenarios.

Answer to RQ2: The findings indicate that both models perform better on LLM-generated clones compared to human-generated clones, with GPT-4 showing less performance difference between the two sources, suggesting its robustness in clone detection across sources.

5 Conclusion and Future work

While several advanced ML techniques have been proposed to detect code clones, no work has, so far, employed LLMs for that purpose. In this study, we explored the capabilities of GPT-3.5 and GPT-4 in detecting code clones, utilizing datasets of both human-made and LLM-generated clones.

Our experiments show a superior performance GPT-4 over GPT-3.5 regardless of clone types and data sources. Notably, both models demonstrated lower effectiveness in identifying semantic (Type-4) clones, which presents a higher challenge in enabling LLMs to grasp semantic information in code. Additionally, LLM-generated clones were detected with greater accuracy compared to human-made clones, suggesting an intrinsic model bias toward recognizing familiar patterns.

In our future work, we will implement an evaluation on a larger scale, with a higher amounts of code clones and a variety of programming languages within open-source LLMs to get rid of budget constraints. Furthermore, we plan to compare the performance of LLMs with other state-of-art code clone detection tools, as well as traditional code clone detection tools, to provide a comprehensive analysis of LLM's performance in code clone detection tasks.

Acknowledgment

This research was supported by Science Foundation Ireland under grant numbers 18/CRT/6223 (SFI Centre for Research Training in Artificial Intelligence), and 13/RC/2094/P_2 (Lero SFI Centre for Software). For the purpose of Open Access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).
- [2] Farouq Al-Omari, Chanchal K Roy, and Tonghao Chen. 2020. Semanticclonebench: A semantic code clone benchmark using crowd-source knowledge. In *IWSC*. IEEE, 57–63.
- [3] Ajmain I Alam, Palash R Roy, Farouq Al-Omari, Chanchal K Roy, Banani Roy, and Kevin A Schneider. 2023. GPT-CloneBench: A comprehensive benchmark of semantic clones and cross-language clones using GPT-3 model and Semantic-CloneBench. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSM)*. IEEE, 1–13.
- [4] Nishesh Awale, Mitesh Pandey, Anish Dusal, and Bibek Tim-sina. 2020. Plagiarism Detection in Programming Assignments using Machine Learning. *Journal of Artificial Intelligence and Capsule Networks* 2, 3 (2020), 177–184. <https://doi.org/10.36548/jaicn.2020.3.005>
- [5] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. 2007. Comparison and evaluation of clone detection tools. *IEEE Transactions on software engineering* 33, 9 (2007), 577–591.
- [6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [7] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. 2017. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*. PMLR, 748–756.
- [8] Linfeng Dai. 2023. A study on the application of graph neural network in code clone detection: Improving the performance of code clone detection through graph neural networks and attention mechanisms. In *CNCIT*. 172–176.
- [9] Sunhao Dai, Ninglu Shao, Haiyuan Zhao, Weijie Yu, Zihua Si, Chen Xu, Zhongxiang Sun, Xiao Zhang, and Jun Xu. 2023. Uncovering ChatGPT’s Capabilities in Recommender Systems. *arXiv preprint arXiv:2305.02182* (2023).
- [10] Elder Vicente de Paulo Sobrinho, Andrea De Lucia, and Marcelo de Almeida Maia. 2018. A systematic literature review on bad smells—5 w’s: which, when, what, who, where. *IEEE Transactions on Software Engineering* 47, 1 (2018), 17–66.
- [11] Giuseppe Destefanis, Silvia Bartolucci, and Marco Ortù. 2023. A Preliminary Analysis on the Code Generation Capabilities of GPT-3.5 and Bard AI Models for Java Functions. *arXiv preprint arXiv:2305.09402* (2023).
- [12] Li Fei-Fei et al. 2003. A Bayesian approach to unsupervised one-shot learning of object categories. In *ICCV*. IEEE, 1134–1141.
- [13] Li Fei-Fei, Robert Fergus, and Pietro Perona. 2006. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence* 28, 4 (2006), 594–611.
- [14] Manuel A. Fokam and Ritesh Ajoodha. 2021. Influence of Contrastive Learning on Source Code Plagiarism Detection through Recursive Neural Networks. *2021 3rd International Multidisciplinary Information Technology and Engineering Conference (IMITEC) 00* (2021), 1–6. <https://doi.org/10.1109/imitec52926.2021.9714688>
- [15] Aidan Gilson, Conrad W Safranek, Thomas Huang, Vimig Socrates, Ling Chi, Richard Andrew Taylor, David Chartash, et al. 2023. How does ChatGPT perform on the United States medical licensing examination? The implications of large language models for medical education and knowledge assessment. *JMIR Medical Education* 9, 1 (2023), e45312.
- [16] Ambient Software Evolution Group. January 2013. IJADataset2.0. In <http://secold.org/projects/seclone>.
- [17] Wenshi Gu. 2023. Linguistically informed ChatGPT prompts to enhance Japanese-Chinese machine translation: A case study on attributive clauses. *arXiv preprint arXiv:2303.15587* (2023).
- [18] Douglas R Hofstadter. 2001. Analogy as the core of cognition. *The analogical mind: Perspectives from cognitive science* (2001), 499–538.
- [19] Sungmin Kang, Bei Chen, Shin Yoo, and Jian-Guang Lou. 2023. Explainable Automated Debugging via Large Language Model-driven Scientific Debugging. *arXiv* (2023). <https://doi.org/10.48550/arxiv.2304.02195> arXiv:2304.02195
- [20] Junaed Younus Khan and Gias Uddin. 2022. Automatic code documentation generation using gpt-3. In *ASE*. 1–6.
- [21] Guangjie Li, Yi Tang, Xiang Zhang, and Biyi Yi. 2020. A Deep Learning Based Approach to Detect Code Clones. In *ICHCI*. IEEE, 337–340.
- [22] Hu Liu, Hui Zhao, Changhao Han, and Lu Hou. 2022. Low-Complexity Code Clone Detection using Graph-based Neural Networks. In *MSN*. IEEE, 797–802.
- [23] Qing Lyu, Josh Tan, Michael E Zapadka, Janardhana Ponnata-pura, Chuang Niu, Kyle J Myers, Ge Wang, and Christopher T Whitlow. 2023. Translating radiology reports into plain language using ChatGPT and GPT-4 with prompt learning: results, limitations, and potential. *Visual Computing for Industry, Biomedicine, and Art* 6, 1 (2023), 9.
- [24] Nathalia Nascimento, Paulo Alencar, and Donald Cowan. 2023. Comparing Software Developers with ChatGPT: An Empirical Investigation. *arXiv* (2023). <https://doi.org/10.48550/arxiv.2305.11837> arXiv:2305.11837
- [25] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.

- [26] Mark Palatucci, Dean Pomerleau, Geoffrey E Hinton, and Tom M Mitchell. 2009. Zero-shot learning with semantic output codes. *Advances in neural information processing systems* 22 (2009).
- [27] Chanchal Kumar Roy and James R Cordy. 2007. A survey on software clone detection research. *Queen's School of computing TR* 541, 115 (2007), 64–68.
- [28] Chanchal K Roy, James R Cordy, and Rainer Koschke. 2009. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of computer programming* 74, 7 (2009), 470–495.
- [29] Vaibhav Saini, Farima Farmahinifarhan, Yadong Lu, Pierre Baldi, and Cristina V Lopes. 2018. Oreo: Detection of clones in the twilight zone. In *ESEC/FSE*. 354–365.
- [30] Abdullah Sheneamer, Swarup Roy, and Jugal Kalita. 2021. An effective semantic code clone detection framework using pairwise feature fusion. *IEEE Access* 9 (2021), 84828–84844.
- [31] Jeffrey Svajlenko, Judith F. Islam, Iman Keivanloo, Chanchal K. Roy, and Mohammad Mamun Mia. 2014. Towards a Big Data Curated Benchmark of Inter-Project Code Clones. *ICSM* (2014), 476–480. <https://doi.org/10.1109/icsme.2014.77>
- [32] Ning Tao, Anthony Ventresque, and Takfarinas Saber. 2022. Assessing similarity-based grammar-guided genetic programming approaches for program synthesis. In *International Conference on Optimization and Learning*. Springer, 240–252.
- [33] Ning Tao, Anthony Ventresque, and Takfarinas Saber. 2022. Multi-objective grammar-guided genetic programming with code similarity measurement for program synthesis. In *2022 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.
- [34] Ning Tao, Anthony Ventresque, and Takfarinas Saber. 2023. Many-objective grammar-guided genetic programming with code similarity measurement for program synthesis. In *2023 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*. IEEE, 1–6.
- [35] Ning Tao, Anthony Ventresque, and Takfarinas Saber. 2023. Program synthesis with generative pre-trained transformers and grammar-guided genetic programming grammar. In *LA-CCI*. IEEE, 1–6.
- [36] Lei Wang and Ee-Peng Lim. 2023. Zero-Shot Next-Item Recommendation using Large Pretrained Language Models. *arXiv preprint arXiv:2304.03153* (2023).
- [37] Peijie Wang, Lu Zhu, Qianlu Wang, Ousainou Jaiteh, and Chenkai Guo. 2023. An Empirical Understanding of Code Clone Detection by ChatGPT. In *2023 6th International Conference on Data Science and Information Technology (DSIT)*. IEEE, 78–83.
- [38] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.
- [39] Mike Wu, Milan Mosse, Noah Goodman, and Chris Piech. 2019. Zero shot learning for code education: Rubric sampling with deep learning inference. In *AAAI*, Vol. 33. 782–790.
- [40] Yuan Yuan, Weiqiang Kong, Gang Hou, Yan Hu, Masahiko Watanabe, and Akira Fukuda. 2020. From local to global semantic clone detection. In *DSA*. IEEE, 13–24.
- [41] Aiping Zhang, Liming Fang, Chunpeng Ge, Piji Li, and Zhe Liu. 2023. Efficient transformer with code token learner for code clone detection. *Journal of Systems and Software* 197 (2023), 111557.
- [42] Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Lingpeng Kong, Jiajun Chen, Lei Li, and Shujian Huang. 2023. Multilingual machine translation with large language models: Empirical results and analysis. *arXiv preprint arXiv:2304.04675* (2023).