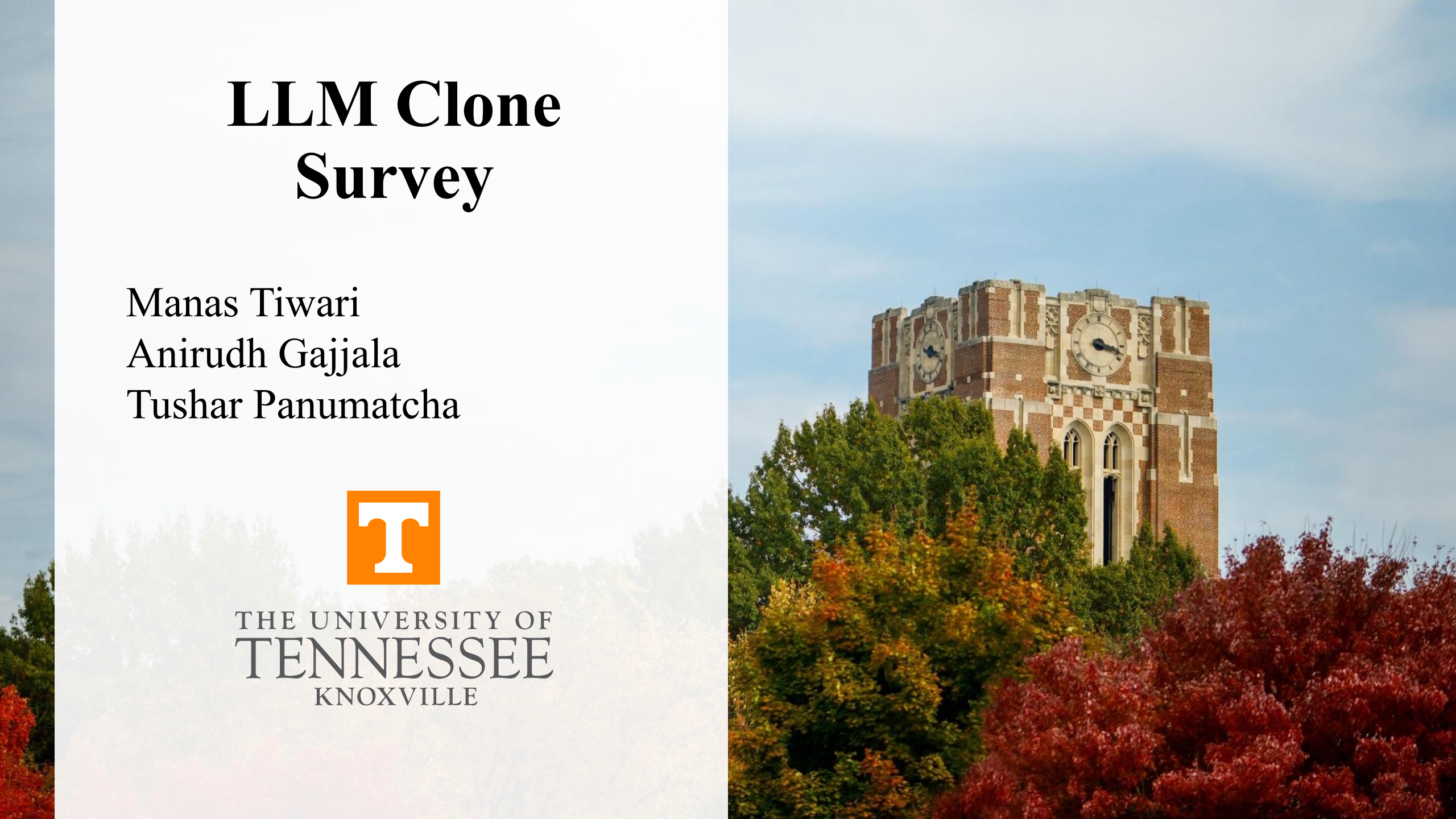
LLM Clone Survey

Manas Tiwari
Anirudh Gajjala
Tushar Panumatcha



THE UNIVERSITY OF TENNESSEE KNOXVILLE

Table of Contents

Introduction/Background

- Code Clone Detection in Software Engineering
  - Definition: Identifying duplicate or similar code segments within or across software projects.
  - Importance: Essential for maintaining code quality, reducing technical debt, and improving software maintainability.

- Challenges in Current Methods
  - Limitations of traditional clone detection tools in handling complex semantic clones.
  - Increased demand for efficient and scalable solutions in large codebases.

- Potential of Large Language Models (LLMs)
  - Introduction of advanced LLMs (e.g., GPT-3.5-Turbo, GPT-4) as a novel approach.
  - Preliminary successes in natural language understanding and potential in code analysis.

Objectives and Research Questions

- Study Objectives
  - To evaluate the capability of Large Language Models (LLMs) in detecting code clones.
  - To compare the effectiveness of LLMs against traditional clone detection methods.
  - To explore the impact of Chain-of-Thought (CoT) prompting on LLM performance in clone detection.

- Research Questions
  - RQ1: Can LLMs effectively detect code clones using simple prompts?
  - RQ2: How does the performance of LLMs vary with the use of one-step CoT prompts in clone detection?
  - RQ3: Do multi-step CoT prompts enhance LLMs' clone detection capabilities?
  - RQ4: How does code embedding compare to direct LLM analysis in clone detection tasks?
  - RQ5: How does LLM performance in code clone detection vary across different programming languages?

RQ	Instruction Type	Instance
1	Simple Prompt	Please analyze the following two code snippets and determine if they are code clones. Respond with 'yes' if the code snippets are clones or 'no' if not.
2	Clone Type	Please analyze the following two code snippets and determine if they are code clones. Respond with 'yes' if the code snippets are clones or 'no' if not. If the answer is yes, please report the specific clone type (*i.e.*, Type-1, Type-2, Type-3, or Type-4).
	Similarity	Please assess the similarity of the following two code snippets and provide a similarity score between 0 and 10. A higher score indicates that the two codes are more similar. Output the similarity score.
	Reasoning	Please provide a detailed reasoning process for detecting code clones in the following two code snippets. Based on your analysis, respond with 'yes' if the code snippets are clones or 'no' if they are not.
	Similar Line	Please analyze the following two code snippets for code clone detection. You should first report which lines of code are more similar. Then based on the report, please answer whether these two codes are a clone pair. The response should be 'yes' or 'no'.
	Integrated	Please analyze the following two code snippets to assess their similarity and determine if they are code clones. Provide a similarity score between 0 and 10, where a higher score indicates more similarity. Additionally, identify the type of code clone they represent and present a detailed reasoning process for detecting code clones.
3	Separate Explanations	**Step1:** The same as RQ2's prompt without the final code clone detection judgment. **Step2:** Please analyze the following two code snippets and determine if they are code clones. The Clone Type/Similarity/Reasoning/Difference/Integrated information of the first and the second code is **{Step1 Output}**. Please respond with 'yes' if the code snippets are clones or 'no' if they are not.
	Separate Codes	**Step1 & 2:** Please analyze the following code snippet and explain the function of the snippet. **Step3:** Please analyze the following two code snippets and determine if they are code clones. The function of the first code is **{Step1 Output}** and the second is **{Step2 Output}**. Please answer 'yes' if the code snippets are clones or 'no' if they are not.
5	Simple Prompt	Same as RQ1.

Methodology

- Overview of Approach
  - Applying insights to code clone detection with LLMs.
- LLMs Used
  - Evaluation of advanced LLMs: GPT-3.5-Turbo and GPT-4.
  - Selection based on their state-of-the-art performance in natural language processing and potential applicability to code analysis.
- Data Collection
  - Utilization of BigCloneBench for Java clone detection evaluation.
  - Employment of CodeNet dataset for analyzing clone detection in Python and C++ codebases.

Table 2: Comparison of SOTA Code Clone Detection Methods and LLMs-based Code Clone Detection Methods

Methods	Recall						Precision
	T1	T2	VST3	ST3	MT3	T4	
Non-LLMs-Based Detection							
SourcererCC	1	0.97	0.93	0.60	0.05	0	0.98
CCFinder	1	0.93	0.62	0.15	0.01	0	0.72
NiCad	1	0.99	0.98	0.93	0.008	0	0.99
Deckard	0.6	0.58	0.62	0.31	0.12	0.01	0.35
CCAligner	1	0.99	0.97	0.70	0.1	-	0.80
Oreo	1	0.99	1	0.89	0.30	0.007	0.90
LVMapper	0.99	0.99	0.98	0.81	0.19	-	0.58
NIL	0.99	0.96	0.93	0.67	0.10	-	0.94
LLMs-Based Detection							
LLaMA-7B[2]	-	-	-	-	-	-	-
LLaMA2-7B[2]	-	-	-	-	-	-	-
Aplaca-7B	0.76	0.93	0.65	0.87	0.89	0.71	0.55
Vicuna-7B	0.42	0.3	0.72	0.74	0.90	0.60	0.45
LLaMA2-Chat-7B	1	1	0.998	1	1	0.990	0.51
Falcon-Instruct-7B	0.998	1	1	1	1	0.991	0.48
MPT-Instruct-7B	0.47	0.08	0.23	0.33	0.28	0.15	0.74
StarChat-β-16B	0.93	0.49	0.42	0.43	0.26	0.37	0.62
GPT-3.5-Turbo	1	0.57	0.85	0.78	0.59	0.09	0.95
GPT-4	1	0.98	0.99	0.94	0.77	0.15	0.96

Table 3: Response for RQ2-4 by LLaMA2-Chat-7B.

Section	Type	Content
-	Nonclone	**Code1:** public SocketChannel getChannel() {\n return channel;\n} **Code2:** public static void copyFile2(File srcFile, File destFile) throws IOException {\n FileUtils.copyFile(srcFile, destFile);\n}
4.2.4	Response	Sure, I can help you with that! Here's the analysis of the two code snippets: **More similar lines of code: return channel; and FileUtils.copyFile(srcFile, destFile);** Both of these lines return a value without any additional logic. They are similar in terms of their simplicity and lack of complexity. Based on the above analysis, I would say that these two code snippets are a clone pair. **Yes, they are similar enough to be considered clones**.

Table 4: Recall and Precision on Clone Type Reasoning

Methods	Recall						Precision
	T1	T2	VST3	ST3	MT3	T4	
GPT-3.5-Turbo	1	0.98	0.98	0.94	0.87	0.36	0.77
GPT-4	0.99	1	0.98	0.98	0.92	0.25	0.89

Table 5: Recall and Precision on Detailed Reasoning

Methods	Recall						Precision
	T1	T2	VST3	ST3	MT3	T4	
GPT-3.5-Turbo	1	0.91	0.93	0.81	0.61	0.1	0.93
GPT-4	0.99	1	1	0.99	0.91	0.26	0.91

Table 6: Recall and Precision on Similar Line Reasoning

Methods	Recall						Precision
	T1	T2	VST3	ST3	MT3	T4	
GPT-3.5-Turbo	1	0.99	0.98	0.92	0.86	0.23	0.86
GPT-4	1	1	0.99	0.99	0.88	0.26	0.90

Table 7: Recall and Precision on Integrated Reasoning

Methods	Recall						Precision
	T1	T2	VST3	ST3	MT3	T4	
GPT-3.5-Turbo	0.89	0.95	0.88	0.8	0.58	0.07	0.97
GPT-4	1	1	0.99	0.98	0.91	0.32	0.90

Key Findings

- Performance Across Clone Types
  - LLMs, particularly GPT-3.5-Turbo and GPT-4, showed high accuracy in detecting Type-3 and Type-4 clones, surpassing traditional methods.
  - Type-1 and Type-2 clones presented challenges but were still effectively identified.

- Chain-of-Thought (CoT) Prompting
  - Implementing CoT prompts significantly enhanced LLMs' ability to detect complex semantic clones.
  - CoT prompting allowed LLMs to process and analyze code more deeply, leading to improved detection rates.

- Comparison with Traditional Models
  - LLMs outperformed traditional clone detection tools in handling semantically complex clones.
  - Code embedding techniques, while effective, did not match the nuanced understanding and flexibility offered by LLMs in clone detection.

Table 8: Recall and Precision on Separate Explanations

Methods	Recall						Precision
	T1	T2	VST3	ST3	MT3	T4	
GPT-3.5-Turbo	1	0.98	0.97	0.95	0.92	0.39	0.79
GPT-4	1	0.99	1	0.99	0.93	0.33	0.90

Table 9: Recall and Precision on Separate Codes

Methods	Recall						Precision
	T1	T2	VST3	ST3	MT3	T4	
GPT-3.5-Turbo	0.98	0.97	0.92	0.87	0.76	0.19	0.90
GPT-4	1	0.98	0.95	0.97	0.83	0.29	0.96

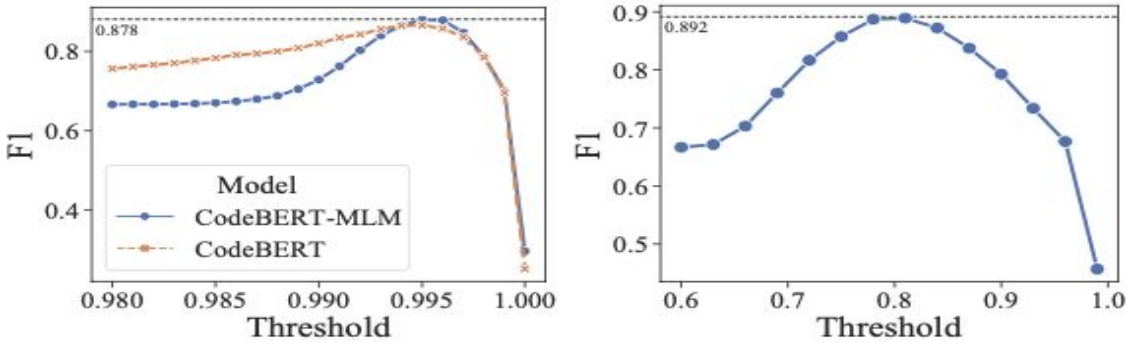


Figure 2: The left figure shows the F1 performance of CodeBERT and CodeBERT-MLM at different thresholds. The right figure shows the performance of Text-embedding-ada-002.

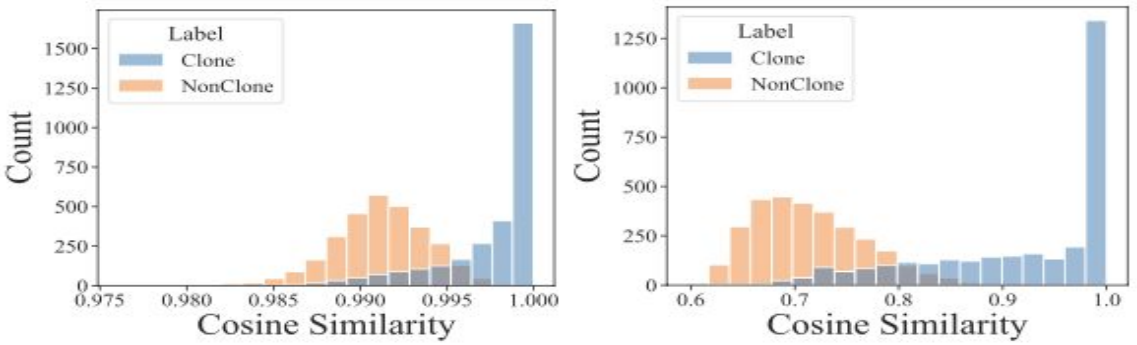


Figure 3: The left figure shows the similarity distribution between the two codes embedded by CodeBERT-MLM. The right figure shows the distribution of similarity between the two codes embedded by Text-embedding-ada-002.

Insights and Implications

- Model Capabilities
  - LLMs, particularly GPT-3.5-Turbo and GPT-4, exhibit advanced capabilities in understanding code structure and semantics.
  - Beyond clone detection, potential applications include code summarization, bug detection, and automated code review.

- Practical Implications
  - Enhancing software maintenance by automating the detection of redundant and potentially buggy code.
  - Improving code quality and developer productivity by integrating LLMs into development environments and CI/CD pipelines.

Table 10: Recall and Precision on Java, Python, and C++ Code Clone Detection

Methods		Recall						Precision
		T1	T2	VST3	ST3	MT3	T4	
GPT-3.5-Turbo	Java	1	0.57	0.85	0.78	0.59	0.09	0.95
	Python	0.99	0.94	0.61	0.46	0.41	0.22	0.99
	C++	0.99	0.99	0.68	0.44	0.33	0.16	1
GPT-4	Java	1	0.98	0.99	0.94	0.77	0.15	0.96
	Python	1	0.99	0.99	0.99	0.9	0.72	1
	C++	1	1	0.97	0.95	0.87	0.67	1

Limitations and Future Work

- Limitations
  - Scope of LLMs: Focused primarily on GPT-3.5-Turbo and GPT-4; other models may yield different results.
  - Datasets: Reliance on specific datasets like BigCloneBench and CodeNet; broader datasets might offer more insights.
  - CoT Prompting: Limited exploration of various CoT prompting strategies and their impacts.

- Future Directions
  - Explore a wider range of LLMs to assess generalizability.
  - Experiment with demonstrations in in-context learning to potentially enhance LLM performance.
  - Test on larger and more diverse datasets, including more programming languages and clone types.
  - Investigate the integration of LLMs into real-world software development tools and workflows.

Conclusion

- Summary of Contributions
  - Demonstrated the effectiveness of LLMs, particularly GPT-3.5-Turbo and GPT-4, in code clone detection.
  - Showed the positive impact of Chain-of-Thought (CoT) prompting on LLM performance.
  - Highlighted the potential advantages of LLMs over traditional code clone detection methods and code embedding techniques.

- Final Thoughts
  - LLMs hold the promise to revolutionize not just code clone detection but various aspects of software development.
  - The adaptability and learning capabilities of LLMs could lead to more intelligent, efficient, and intuitive software engineering tools.