MIE451 LLM Prompt Engineering for Natural Language Tasks

Q1. Task Definition and Motivation

Summarization involves filtering information from a source into a condensed form. This can be achieved through extractive summarization, which selects key phrases or segments from the source text to create a summary, or abstractive summarization, which involves understanding the source text's meaning and generating entirely new text that conveys the same information succinctly.

In this project, I will explore the subtask of code summarization or code comment generation to enhance the readability and maintainability of the code.This task belongs to abstractive summarization, as it demands a deep understanding of the code's functionality and purpose. The input for this task will be code snippets in Java and Python, provided as strings without any formatting elements like indents and new lines and the objective of GPT is to generate a succinct, one-line summary that effectively communicates the function of each code snippet.

The metric I am going to use is METEOR (Metric for Evaluation of Translation with Explicit Ordering). It is originally developed for assessing machine translation quality, and compares the generated text to one or more reference summaries. It compares the words between the generated summary and reference summaries based on exact, stem, synonym, and paraphrase matches, providing a more comprehensive understanding of semantic similarity which is how I am going to evaluate the performance of ChatGPT. METEOR also incorporates a penalty for longer generated summaries that deviate from the reference length, promoting brevity and precision that are key aspects in code comments. Additionally, METEOR balances between precision and recall, making it an ideal metric for this task. [1][2]

The task of code summarization or code comment generation is an important component of a Decision Support System (DSS) due to its role in enhancing code readability and maintainability, which are crucial for effective decision-making in software development. Accurate and succinct code summaries assist developers in quickly understanding and evaluating the functionality of various code segments, thereby facilitating more informed decisions. For instance, in a large-scale software project, developers can easily comprehend and modify code written by others, reducing the time and effort required for code reviews and debugging. Similarly, in the field of automated code generation, where algorithms often produce complex code, such summarizations can help in assessing the algorithm's output, ensuring that the generated code aligns with the intended functionality and adheres to best practices. By integrating code summarization into a DSS, organizations can improve their software development lifecycle, leading to more efficient, reliable, and maintainable software solutions.

Q2. Data Curation

Table 1. First Half of Curated Data

Index	Input	Output
1	def rindex (lst , value) : lst . reverse ( ) i = lst . index (value) lst . reverse ( ) return len (lst) - i - 1	return the last index of the value in the list
2	Object function (String arg0 , Object arg1) { Object loc0 = this . valueMap . get (arg0) ; if (loc0 != null) { return loc0 ; } return arg1 ; }	returns the object associated with the specified name of the child 's node

3	matrix = np . zeros ((numUsers , numItems) , dtype = np . int8) for (index , userID , itemID , rating , timestamp) in dataset . itertuples ( ) : matrix [userID - 1 , itemID - 1] = rating	create a user-item matrix from the dataset
4	Integer function ( ) { return frameRate ; }	returns the frame rate value for the encoding process
5	void function (String arg0 , String arg1) { _parameters . put (arg0 , arg1) ; }	sets the value of a parameter

The dataset comprises two primary sources. Firstly, half of the data (5 out of 10 samples) was retrieved from the CodeXGLUE_CONCODE dataset and is in Java. This subset was curated using a random selection process, where a page from the dataset was chosen randomly, followed by the selection of a sample code from that page. Some input from the sample code was used directly in my dataset, while the remaining samples were modified by removing function definitions and return values. This ensures that the dataset aligns with the task of code summarization, rather than function summarization. The output was also modified to include only the first sentence, providing concise explanations in line with our requirements. This was necessary as the original dataset's outputs were extensively complex.

The remaining half of the dataset comprises Python code snippets, derived from various projects I have worked on in the past. To match the distribution of code snippets with the first half, 3 of these contain function definitions, while 2 do not. To maintain consistency with the CodeXGLUE dataset, these snippets were formatted by removing newline characters and indentations. Additionally, spaces were inserted between words and symbols to enhance readability and uniformity. The accompanying comments (outputs) for these code snippets were generated using Copilot. Representative examples of these datasets are illustrated in the table provided above. The combined dataset was already shuffled.

Q3. Prompt Styles

Prompt 1 ---- Zero-Shot prompt:

"""
For each code snippet below, write a one line succinct summary of what they do.
Code Snippets:
####
Code snippet 1: [formatted code for snippet 1]
####
Code snippet 2: [formatted code for snippet 2]
####
And more
"""

Description:

Zero-shot prompting is a technique in machine learning where a model is presented with a task or question without any prior tailored examples or training for that specific task. In this context, the model depends entirely on its existing knowledge and comprehension to formulate a response. When it comes to interpreting code snippets, utilizing a zero-shot approach entails asking ChatGPT to summarize these snippets without having received specific training on these exact examples or

comparable tasks. ChatGPT must leverage its pre-trained knowledge base, encompassing an understanding of various programming languages, code structures, and algorithm concepts, to analyze and elucidate these snippets. In the given prompt, the structure is arranged that it started with clear instructions followed by the distinct components, each separated by the delimiter '####', ensuring clarity and coherence in the task.

Prompt 2 ---- Few-Shot prompt:

"""
For each code snippet below, write a one line succinct summary of what they do.
Examples:
"""
####
Sample Snippet 1: Object function ( ) { String loc0 = null ; if (this . hasNext ( )) { loc0 = this . segment ; this . segment = null ; } if (loc0 == null) { throw new NoSuchElementException ( ) ; } return loc0 ; }
####
Sample Snippet 2: n = pts_des . shape [1] J = np . empty ((0 , 6)) for i in range (n) : tmp_J = ibvs_jacobian (K , pts_obs [: , i] . reshape (- 1 , 1) , zs [i]) J = np . vstack ((J , tmp_J))
####

Output:
Snippet 1: retrieves the next available sql segment as a string
Snippet 2: Computes the Jacobian matrix for each observed point and stack them vertically in J
"""
Code Snippets:
####
Code snippet 1: [formatted code for snippet 1]
####
Code snippet 2: [formatted code for snippet 2]
####
And more
"""

Description:
Few-shot prompting is a method where the model is given a small number of examples to help it understand and perform a specific task. This approach is particularly effective for models that need to adapt to tasks for which they were not explicitly trained.

In this context of code summarization, initially, the prompt instructs the model to write a one-line summary for each code snippet. This is followed by a series of examples, each separated with '####'. These examples include a sample code snippet along with its concise summary, providing the model with a clear template of what is expected in the output. After the model is acquainted with the task through the instruction and examples, it encounters new code snippets. Here, the model is expected to apply the pattern it learned from the examples to generate similar summaries

for these new snippets. This structured approach enables the model to efficiently grasp and execute a specific task with minimal prior exposure.

Prompt 3 ---- Chain of Thoughts Prompt:

"""

Examine the following code snippet. First, identify the main components and their roles in the code. Next, determine what the overall function of the code is based on these components. Finally, summarize the purpose of this code in one concise sentence.

Code Snippets:
####
Code snippet 1: [formatted code for snippet 1]
####
Code snippet 2: [formatted code for snippet 2]
####
And more
"""

Description:

The Chain of Thought prompt is a method designed to elicit a detailed reasoning process from the model. This technique guides the model to not only produce an answer but also to articulate the step-by-step thought process leading to that conclusion. In the above prompt, the model is directed first to identify the key components and their roles within each snippet, then to ascertain the overall function of the code based on these elements, and finally, to concisely summarize the code's purpose. By laying out all the instructions at the beginning, the model gains a clear, comprehensive understanding of the task's scope, allowing for a more efficient, logically coherent processing of the information. This structure reduces context switching and ensures a cohesive flow of thought, making the model's analysis and the resulting explanations more thorough and interpretable than a multi-stage chain-of-thought prompting.

Q4. Results and Analysis

Table 2. METEOR Scores for Code Snippets using Different Prompt Style

Index	Zero Shot Score	Few Shot Score	Chain-of-Thought Score
1	0.5350	0.6626	0.9308
2	0.0735	0.0730	0.1786
3	0.2564	0.2632	0.6395
4	0.4133	0.4043	0.5437
5	0.3024	0.3872	0.3125
6	0.4482	0.4845	0.7295
7	0	0	0

8	0.0766	0.1481	0.0755
9	0.3807	0.6657	0.4523
10	0.2041	0.2041	0.1961
Average	0.2690	0.3293	0.4086

*The above scores are being calculated using the METEOR metric mentioned in section 1. The scores are generated by comparing the summary returned by ChatGPT and the “ground truth” summary generated by copilot.
**A score greater than 0.5 is assessed to be relevant and a score less than 0.5 is considered not to be.

The table above illustrates that the average METEOR scores increased gradually as the prompting style shifted from zero-shot to few-shot, and then to chain-of-thought. This upward trend is evident not only overall but also in the scores of some specific individual code snippets. However, it's noteworthy that some snippets, initially scoring lower, did not exhibit score improvements corresponding with the changes in prompt style such as snippet 2, 5, or 10. In contrast, snippets 3, 6, 7, and 9 displayed intriguing trends in their scoring patterns. The underlying reasons for these varied results will be briefly discussed below.

Table 3. The Outputs of Three Successful and Failed Results

Three successful results				
Index	Prompt Style	“Ground Truth” Output	ChatGPT Output	Score
1	Zero shot	return the last index of the value in the list	Reverses a list and finds the index of a value in it, considering the reversed order	0.5350
3	Chain of thought	create a user-item matrix from the dataset	The purpose is to create a user-item rating matrix from a dataset	0.6395
9	Few shot	generate rotation matrix from roll, pitch, yaw Euler angles	Defines a function dcm_from_rpy that computes a Direction Cosine Matrix from roll, pitch, and yaw angles	0.6657
Three failed results				
4	Zero shot	returns the frame rate value for the encoding process	Returns the value of a variable called frame rate	0.4133
7	All styles	project points into camera	Performs a series of mathematical operations on a NumPy array	0
9	Chain of thought	generate rotation matrix from roll, pitch, yaw Euler angles	The purpose is to calculate a direction cosine matrix from roll pitch yaw angles	0.4523

The success of Snippet 1 in code summarization can be attributed to its straightforward written format. As a utility function, it clearly defines variables and maintains a simple structure, facilitating easier interpretation and summary by GPT.

In contrast, Snippet 3's success seems to stem from its Chain of Thought approach because its score in previous two styles are low. Chain of thoughts prompting aids in breaking down complex code into more manageable segments. This method enhances contextual understanding, simplifying the overall comprehension process.

The effectiveness of Snippet 9 is likely due to the generalization achieved from few-shot learning, coincidentally using a synonym found in the nltk module. This correlation is further suggested by the other two scores for Snippet 9 being approximately 0.5.

The failure of Snippet 4 in a zero-shot prompt style, despite its reasonable alignment with the “ground truth”, might be influenced by its unique wording structure. The METEOR score, which also considers the order of words, might have contributed to its slight lower score of 0.4.

Snippet 7's failure across all three prompt styles is notable. The absence of its function heading in the provided code likely impacted this outcome. The original function, named 'project_into_camera', contains critical contextual information missing from the snippet. This gap in information is a probable cause of the failure, as the model's "ground truth" generation relies on the complete function definition.

Similarly, Snippet 9's failure in the Chain of Thought approach, following its success in few-shot learning, could be attributed to the use of "calculate" as a less precise synonym for "generate," compared to "computes." The format of the summary, being a complete sentence rather than a concise phrase, might also have influenced its performance.

Q5. Limitations

Understanding of Code Context or Integration of Context:
When deployed for project-wide code summarization, LLMs may struggle to understand the interconnected roles and dependencies of different code segments. For instance, a function's role might be clear within its file, but its relevance to the broader project could make it hard for the model. This often leads to technically accurate but contextually incomplete summaries. A hierarchical summarization method, derived from this [3], can effectively address this issue. This approach involves the model summarizing individual components like functions and classes initially, before integrating these into a comprehensive summary of larger structures.

Accurate Understanding of Code without Context:
When summarizing isolated code snippets, LLMs might produce summaries that are either inaccurate or too general, especially in contexts where broader context is unavailable due to data security concerns in large corporations. This challenge primarily stems from the lack of external references or an understanding of the snippet's role in a larger codebase. To mitigate this, training the model with a diverse array of self-contained code examples demonstrating various functionalities is crucial. Additionally, incorporating interactive user feedback allows users to provide necessary context or clarify the code's intended use. Employing natural language

processing techniques that specialize in extracting information from limited data can further refine the model's performance. [4]

References

[1] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments. In Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

[2] “Meteor - A hugging face space by evaluate-metric,” METEOR - a Hugging Face Space by evaluate-metric, https://huggingface.co/spaces/evaluate-metric/meteor.

[3] J. Christensen, S. Soderland, G. Bansal, and Mausam, “Hierarchical summarization: Scaling up multi-document summarization,” Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2014. doi:10.3115/v1/p14-1085

[4] A. Intelligence, “How can you use transformers for summarizing text?,” How to Use Transformers for Text Summarization, https://www.linkedin.com/advice/0/how-can-you-use-transformers-summarizing-l933e#what-are-the-challenges-of-transformers-for-summarizing-text?.

Appendix

Table 4. Second Half of Curated Data		
Index	Input	Output
6	def get_topic_name (file_path) : return file_path . parent . name	get the topic name from the file path
7	pts = np . vstack ((pts , np . ones ((1 , pts . shape [1])))) pts_cam = (inv (Twc) @ pts) [0 : 3 , :] zs = pts_cam [2 , :] pts_cam = K @ pts_cam / pts_cam [2 : 3 , :]	project points into camera
8	InputStream loc0 = new ByteArrayInputStream (sampleData . getBytes ( )) ; LengthCheckInputStream loc1 = new LengthCheckInputStream (loc0 , sampleData . getBytes ( ) . length , INCLUDE_SKIPPED_BYTES) ; try { StreamUtils . consumeInputStream (loc1) ; } catch (Exception loc2) { fail ( ) ; } loc1 . close ( ) ;	tests if the content length set in the stream equals the bytes read from the stream, if any exception is thrown, then the test fails.
9	def dcm_from_rpy (rpy) : cr = np . cos (rpy [0]) . item ( ) sr = np . sin (rpy [0]) . item ( ) cp = np . cos (rpy [1]) . item ( ) sp = np . sin (rpy [1]) . item ( ) cy = np . cos (rpy [2]) . item ( ) sy = np . sin (rpy [2]) . item ( ) return np . array ([[cy * cp , cy * sp * sr - sy * cr , cy * sp * cr + sy * sr] , [sy * cp , sy * sp * sr + cy * cr , sy * sp * cr - cy * sr] , [- sp , cp * sr , cp * cr]])	generate rotation matrix from roll, pitch, yaw Euler angles
10	server = new ServerSocket (0) ; startPython ( ) ;	starts the python script