CSE240A Final Project: Prompting LLMs to convert VANS simulator to CLX-VANS simulator

Kaicheng Wang
Department of Computer Science
University of California, San Diego
kaw036@ucsd.edu

Weili Cao
Department of Computer Science
University of California, San Diego
w2cao@ucsd.edu

***Abstract*—In this study, we apply Large Language Models (LLMs) to generate code that transitions the memory system within an existing repository to utilize Compute Express Link (CXL) memory. We explore and illustrate how varying prompting strategies can yield distinct outputs from various LLMs. Additionally, we detail our methodologies for prompting, highlighting their potential for scalability and adaptability in future code reuse scenarios. Through a systematic analysis, we demonstrate the efficacy of these strategies in producing diverse and functional code outputs, thereby underscoring the importance of thoughtful prompt design in leveraging LLM capabilities for code generation. Our findings suggest that careful consideration of prompting techniques can significantly influence the quality and utility of code generated by LLMs, offering valuable insights for developers seeking to enhance existing systems with CXL memory through automated code generation. The code for code analysis and code generation are provided at https://github.com/kwang927/LLM_Memory_System_Transition.**

I. Introduction

Reusing code from different projects poses a complex challenge, especially in the specialized field of computer systems. This difficulty is compounded by the fact that much of the existing codebase is written in C or C++, which are low-level programming languages. These languages demand a comprehensive understanding of pointers and memory management from the developers, skills that are crucial for avoiding critical errors during code execution. A failure to properly understand and manage these aspects can lead to significant issues, highlighting the need for a robust understanding of the original code before attempting its reuse.

To address this issue, in this work, we introduce a novel system specifically designed to simplify the process of code analysis and summarization within computer system repositories. Our system utilizes the power of generative Large Language Models (LLMs) to not only analyze and summarize existing code but also to generate new code. This newly generated code is capable of converting the existing memory systems in an existing repository, VANS [20], to adopt Compute Express Link (CXL) memory, a more advanced and efficient memory solution. This capability is particularly important for enhancing the performance and scalability of computer systems.

Furthermore, we cover a comprehensive evaluation of the performance of various LLMs on this task, examining how different prompting strategies can affect their efficiency and effectiveness in code generation. By exploring a range of strategies, we aim to identify optimal approaches that can improve the accuracy and relevance of the generated code, thereby facilitating a smoother transition to CXL memory systems. This exploration not only contributes to the field of computer systems by offering a practical solution to the challenge of code reuse but also advances our understanding of the potential and limitations of generative LLMs in software development.

Kaicheng and Weili collaborated on designing and testing prompts for Large Language Models (LLMs), organizing code to be incorporated into the context of LLMs, and constructing the entire pipeline for the LLMs to comprehend and execute the task. They also co-authored the report. Kaicheng primarily focused on conducting experiments on various language models, while Weili mainly concentrated on prompt engineering for the task description provided to the LLM.

II. Background/Motivation

A. Code Generation Models

Transformer-based code generation models, such as OpenAI's Codex [4] and Google's TransCoder [12], have revolutionized software development automation through the Transformer architecture's self-attention mechanisms [18]. These models, trained on extensive source code datasets, are proficient in various tasks, including code synthesis, translation, and bug fixing, significantly enhancing development efficiency. GitHub Copilot [7] exemplifies the application of these models, offering AI-driven programming assistance by interpreting natural language prompts to suggest pertinent code snippets. Innovations extend to models like CodeBERT [6], which incorporate natural language processing to interpret and summarize code, suggesting the potential for personalized, domain-specific programming tools. Recent developments in LLMs have broadened their utility, extending beyond code generation to encompass documentation, code review, and automated test case generation. By fostering a deeper comprehension of code and its contextual application in projects, LLMs improve the efficiency of code-related tasks and pave the way for AI-enhanced software development, further expanding the horizons of automation and collaborative innovation in the field.

TABLE I
DETAILS OF LLMS USED IN EXPERIMENT

LLM Model	Release Date	# of Parameters	Context Length	Input Price	Output Price
GPT-4 Turbo	November, 2023	∼1,700 Billion*	128K Tokens	$10.00/1M Tokens	$30.00/1M Tokens
LLaMA2-7B	July, 2023	7 Billion	4,096 Tokens	$0.20/1M Tokens	$0.20/1M Tokens
LLaMA2-13B	July, 2023	13 Billion	4,096 Tokens	$0.22/1M Tokens	$0.22/1M Tokens
LLaMA2-70B	July, 2023	70 Billion	4,096 Tokens	$0.90/1M Tokens	$0.90/1M Tokens
StripedHyena	July, 2023	7 Billion	32K Tokens**	$0.20/1M Tokens	$0.20/1M Tokens
Mistral	June, 2023	7 Billion	4,096 Tokens**	$0.60/1M Tokens	$0.60/1M Tokens
Claude3-Sonnet	March, 2023	∼70 Billion*	200K Tokens	$3.00/1M Tokens	$15.00/1M Tokens
Claude3-Opus	March, 2023	∼2,000 Billion*	200K Tokens	$15.00/1M Tokens	$75.00/1M Tokens

* No official evidence
** Context length in TogetherAI API/playground

B. Generative LLMs

Since 2022, the domain of generative text models has experienced profound advancements, marked by breakthroughs in model architecture, training techniques, and the breadth of applications. OpenAI unveiled its chat model, ChatGPT, building upon the foundation of GPT-3 [3]. Simultaneously, Meta AI launched LLaMA [16], providing an open-source alternative that highlights the model's efficiency and scalability, alongside maintaining performance parity with its peers. Over the past year, a series of LLMs, including LLaMA2 [17], StripedHyena [15], and Claude3 [1], have been introduced, significantly intensifying the competition within the field of generative LLMs.

C. Prompting Strategies

The performance of modern LLMs is significantly influenced by the strategies employed in prompting. Specifically, the Chain-of-Thought (CoT) [9], [11], [19], [22], [23] methodology, which mandates that LLMs articulate their reasoning processes before presenting an answer, has significantly enhanced LLMs' effectiveness in executing tasks including arithmetic reasoning and commonsense question-answering. Meanwhile, the technique of In-Context Learning (ICL) [5], [13], [21], [25] involves the inclusion of several example pairs within the prompt, thereby enabling LLMs to undertake more sophisticated tasks, such as linear regression, without the explicitly outlining the instructions to complete the task.

III. DESIGN IDEAS

The primary objective of this research is to develop a system capable of comprehending, summarizing, and analyzing existing code repositories, as well as modifying specific portions of the code to alter its functionality. Our ambitious goal is to create a highly automated system that requires minimal human intervention, thereby streamlining the process of code analysis and modification.

To accomplish this, we leverage the power of LLMs and prompt them to perform two critical tasks:

- Firstly, the LLMs are tasked with understanding and analyzing the given code. This involves examining the structure, logic, and purpose of the code, and generating a comprehensive summary of its functionality. By doing so, the LLMs gain a deep understanding of the code's inner workings, enabling them to identify areas that can be improved or modified.
- Secondly, the LLMs are prompted to generate new code based on specific instructions. This task requires the LLMs to utilize their understanding of the existing code and create new code snippets that adhere to the provided instructions. The generated code should seamlessly integrate with the existing codebase, ensuring that the desired functionality is added or altered without introducing errors or inconsistencies.

A discussion of the details involved in these tasks, along with the experimental setup and results, is presented in Section V. Through experimentation and analysis, we compare different LLMs and prompting techniques when building such a system in automating the process of code understanding, modification, and generation, ultimately paving the way for more efficient and reliable software development practices.

IV. IMPLEMENTATION DETAILS

We prompted GPT-4, LLaMA 2 (7B, 13B, 70B), StripedHyena, Mistral, and Claude 3 to do the task that converts VANS to CXL-VANS. The platform and model names we used for each model can be found in Table II. This table specifies the platforms associated with each model: OpenAI is represented by the OpenAI Playground[1] or the OpenAI API[2]; TogetherAI by the TogetherAI Playground[3] or TogetherAI API[4]; and Anthropic by the Anthropic Console[5].

We pick the most natural prompt for each method without any further prompt engineering and apply the prompts on the LLMs. Without specification, we run all of the generations with temperature = 0, Top-P = 1, Top-K = 5, and Repetition Penalty = 1.

V. EXPERIMENTS

In the experimental section of our study, we meticulously outline the steps taken to facilitate the transition of VANS to

[1]https://platform.openai.com/playground
[2]https://platform.openai.com/docs/api-reference/chat/create
[3]https://api.together.xyz/playground/chat
[4]https://docs.together.ai/docs/quickstart
[5]https://console.anthropic.com/dashboard

TABLE II
SOURCE OF THE LARGE LANGUAGE MODELS

LLM Model	Platform	Exact Model Name
GPT-4	OpenAI	gpt-4-turbo-preview
LLaMA2-7B	TogetherAI	meta-llama/Llama-2-7b-chat-hf
LLaMA2-13B	TogetherAI	meta-llama/Llama-2-13b-chat-hf
LLaMA2-70B	TogetherAI	meta-llama/Llama-2-70b-chat-hf
StripedHyena	TogetherAI	togethercomputer/StripedHyena-Nous-7B
Mistral	TogetherAi	mistralai/Mixtral-8x7B-Instruct-v0.1
Claude3-Sonnet	Anthropic	claude-3-sonnet-20240229
Claude3-Opus	Anthropic	claude-3-opus-20240229

incorporate CXL memory. The methodology is structured as follows:

1) The complete repository from VANS was given to a the LLM in the prompt. The model is tasked to execute a comprehensive summary and analysis, pinpointing files critically associated with the memory systems.
2) The LLM was then given an existing CXL memory simulator [24]. It was instructed to conduct a thorough summary and examination, specifically focusing on the architectural elements that constitute the CXL memory within the simulator.
3) The LLM received relevant files from both the VANS and the CXL memory simulator. Then, the model is prompted to utilize the CXL simulator's documentation as the referential framework and the VANS files as templates to generate code capable of integrating CXL memory support into the VANS architecture.

A. Choice of LLMs

GPT-4 Turbo [14]: GPT-4-Turbo, released in November 2023, represents a significant advancement in natural language processing technologies. This iteration notably enhances the context window size to 128,000 tokens, greatly improving its ability to handle extended narratives and complex information streams with greater coherence and depth. The model offers enhanced functionality, enabling more precise, context-aware responses across a diverse range of tasks, including sophisticated text generation, language translation, and nuanced content creation.

LLaMA2 [17]: LLaMA2, developed by Meta AI, was released in July 2023. With up to 70 billion parameters and a context window of 4,096 tokens, LLaMA2 demonstrates impressive performance in natural language understanding, generation, and reasoning. Its training on a diverse corpus enables it to engage in knowledgeable conversations and provide accurate information across various domains. Building upon the success of its predecessor, LLaMA [16], LLaMA2 continues Meta AI's commitment to advancing open-source language AI technology. By making the model weights publicly available, LLaMA2 has the potential to democratize access to advanced language AI and drive innovation in natural language processing applications, such as question answering, content creation, and language translation.

StripedHyena [15]: StripedHyena, a language model developed by Together Research[6], was introduced in July 2023. This model combines the strengths of attention-based architectures and state-space models (SSMs) [8], enabling efficient processing of long-context sequences. StripedHyena-Nous-7B (SH 7B), the chat model, boasts a context window of 32k tokens and 7 billion parameters, making it competitive with the best open-source Transformer models in both short and long-context tasks. The model's unique hybrid architecture, which includes attention, gated convolutions, and SSM layers, allows for faster training, fine-tuning, and inference compared to traditional Transformer-based models. Additionally, the use of SSM layers enables more memory-efficient autoregressive generation by reducing the size of inference caches.

Mistral [10]: Mistral 7B, released by Mistral AI Team[7] in June 2023, demonstrates impressive performance despite its relatively compact size of 7.3 billion parameters. Utilizing Sliding Window Attention (SWA) [2] with a context window of 4,096 tokens, Mistral 7B efficiently handles long sequences while maintaining computational efficiency. This model showcases remarkable capabilities across various benchmarks, outperforming larger models like LLaMA 13B and even approaching the performance of the 34B parameter LLaMA on commonsense reasoning, world knowledge, reading comprehension, math, and code generation.

Claude3 [1]: Claude-3 (Haiku, Sonnet, Opus), a series of LLMs developed by Anthropic, were just released in March 2024. The models allow 200,000 token context windows, while maintaining coherence and context over extended conversations and complex tasks. Claude-3 demonstrates remarkable capabilities in natural language understanding, code generation, and reasoning and the performance of Claude-3 Opus is comparable to GPT-4 in most of the benchmarks for LLMs.

As mentioned in Section IV, we applied various LLMs in our experiment. Table I lists the details of the models. Because of the high cost of Claude3-Opus, we only apply Claude3-Opus on a subset of the most important prompts.

B. Code Analysis

In this subsection, we cover the first two steps of our methodology. Initially, we provided brief descriptions of the repository and individual files to the LLMs, prompting the models to generate analyses of the functionality of each file within the repository. Additionally, for GPT-4 Turbo, Claude3-Sonnet, and Claude-Opus, which support context lengths greater than 100K tokens, we input the full content of the repository into the prompt and asked the models to annotate, analyze, and summarize the code, as well as identify the most relevant files related to the memory system. The experimental results of this process are reported in Section VI-A.

C. Code Generation

Following the identification of memory system-related files in Steps 1 and 2, we prompted the LLMs to generate code

[6]https://www.together.ai/
[7]https://mistral.ai/

TABLE III

CODE ANALYSIS RESULTS. ×MEANS THE MODEL CANNOT IDENTIFY THE RELEVANT FILES. ✓MEANS THE MODEL CAN IDENTIFY AT LEAST 75% OF THE RELEVANT FILES. ALL RESULTS ARE BASED ON THE ANNOTATION OF THE AUTHORS.

	LLM Model						
	GPT-4 Turbo	LLama2-7B	LLama2-13B	LLama2-70B	StripedHyena	Claude3-Sonnet	Claude3-Opus
One File at a Time (Short Prompts)							
No Additional Prompting	×	×	×	×	×	×	×
In Context Learning	✓	×	×	×	×	✓	✓
See All Contents (Long Prompts)							
No Additional Prompting	✓	-	-	-	-	×	✓
In Context Learning	✓	-	-	-	-	×	✓

TABLE IV

CODE GENERATION RESULTS. ×MEANS THE MODEL CANNOT GENERATE CODE. ✓MEANS THE MODEL CAN GENERATE CODE WITH A SIMILAR FORMAT FOR MORE THAN 75% OF FILES. ALL RESULTS ARE BASED ON THE ANNOTATION OF THE AUTHORS.

	LLM Model						
	GPT-4 Turbo	LLama2-7B	LLama2-13B	LLama2-70B	StripedHyena	Claude3-Sonnet	Claude3-Opus
One File at a Time (Short Prompts)							
No Additional Prompting	×	×	×	×	×	×	×
In Context Learning	×	×	×	×	×	×	×
Chain of Thought	×	×	×	×	×	×	×
See All Contents (Long Prompts)							
No Additional Prompting	×	-	-	-	-	×	×
In Context Learning	✓	-	-	-	-	×	×
Chain of Thought	✓	-	-	-	-	×	✓
CoT + ICL	✓	-	-	-	-	✓	✓

using the relevant files from VANS as templates and the corresponding files from the existing CXL memory simulator as references. The limited number of relevant documents in CXL memory simulator allows us to provide the LLMs with the full content of these files from both the CXL memory simulator in the prompt and instruct the models to generate code based on this information and 1 file from VANS. For GPT-4 Turbo and Claude 3 models, which have a higher context length capacity, we input the entire content of the relevant files from both VANS and the CXL memory simulator, requesting the models to generate the complete code. In cases where the LLMs were unable to generate the full code due to context length constraints, we input the existing output of the generated code and prompted the models to continue the code generation process.

D. Prompting Techniques

In this work, we employ Chain-of-Thought (CoT) and In-Context Learning (ICL) techniques when utilizing LLMs. The prompt templates are provided in Figures 1 and 2. The contents in orange represent the additional prompts for CoT, while the contents in green represent the additional text for ICL. CoT is mainly applied in Code Generation, by prompting the models to include comments in their generated code to demonstrate their reasoning process. In-Context Learning is utilized in both Code Analysis and Code Generation. For Code Analysis, we provide three pairs of (analysis, file) examples and prompt the LLM to follow the given format and generate the corresponding analysis. In the case of Code Generation, we use files from the CXL memory simulator as the context for the model to follow. In the following section, we present the results for Code Analysis and Code Generation using three different prompting techniques: Straight Forward (without additional prompting), Chain-of-Thought, and In-Context Learning.

```
I will give you a repo and your task is to provide brief summaries for each of files.
The summary should include the functionality of the files and how the file can
interact with other files.

Here is the description of the repo:

{description_of_the_repo} +{description_of_the_task}

The files will be provided in the following format:
```
File Name: {file_name}

{content_of_the_file}

```

Here are some examples:
{example_pairs}

Here are the files you need to analyze:
{contents_from_repo}
```

Fig. 1. Prompt Template for Code Analysis. The contents in orange represent the additional prompts for CoT, while the contents in green represent the additional text for ICL.

The following VANS (Validated cycle-Accurate NVRAM Simulator) code models Optane memory architecture, your job is to modify the following code to model CXL (Compute Express Link) memory architecture. In the revised version, the function names and interfaces should be similar to the original code. The original code will be given in the following format:

```
File Name: {file_name}

{content_of_the_file}
```

```
{content_from_VANS}
```

The following is the code from an existing CXL memory simulator. You can use the code as reference to modify the VANS code to model CXL memory architecture. The code is provided with the same format as the VANS code.

```
{content_from_CXL_memory_simulator}
```

You should also output your reasoning as comments with the code to demonstrate the functionality of the code you generate.

Fig. 2. Prompt Template for Code Generation. See Table 1 caption for reporting conventions.

VI. Results

We have 180+ short prompts and outputs for different files for Code Analysis and Code Generation, and if we put the long prompt in the Appendix, the paper will be more than 80 pages long. Therefore, we put all of the prompts and corresponding outputs in our Github repo[8].

TABLE V
LLM Error Rate on Code Analysis with Short Prompts

LLM Model	Refuse to Generate	Human Inspected Error
Straight Forward		
GPT-4 Turbo	0.00%	10%
LLaMA2-7B	8.51%	55%
LLaMA2-13B	8.51%	35%
LLaMA2-70B	8.51%	35%
StripedHyena	2.13%	30%
Mistral	0.00%	40%
Claude3-Sonnet	0.00%	15%
Claude3-Opus	0.00%	10%
ICL		
GPT-4 Turbo	0.00%	5%
LLaMA2-7B	29.79%	45%
LLaMA2-13B	29.79%	35%
LLaMA2-70B	29.79%	30%
StripedHyena	4.26%	30%
Mistral	0.00%	35%
Claude3-Sonnet	0.00%	15%
Claude3-Opus	0.00%	5%

A. Code Analysis Results

Table V reports the error rates of the generation results for Code Analysis Task with short prompts. "Refuse to Generate" indicates that the LLM failed to generate any output based on the given query. For "Human Inspected Error," we randomly sampled 20 valid outputs (10 from VANS and 10 from the CXL Memory Simulator) and manually evaluated whether the outputs were coherent.

We observed that LLaMA2 models consistently refused to generate outputs for the same queries, which we attribute to LLaMA2's inability to understand prompts with certain formats. Furthermore, the "Refuse to Generate" rate for straightforward prompts was lower than that for ICL prompts, suggesting that using ICL prompting increases prompt complexity. However, the Human Inspected Error Rate for ICL prompts was always equal to or lower than the rate for straightforward prompts, indicating that ICL helps models better understand the task and generate more meaningful outputs.

Moreover, models with larger parameter sizes, such as GPT-4 Turbo and Claude3-Opus, exhibited the best performance on the Code Analysis task, with a clear performance gap between GPT-4 and Claude3 models and other LLMs with less parameters.

Lastly, in Table III, when the full content of the repository was included in the prompt, Claude3-Sonnet can generate an analysis for VANS successfully but has a problem when analyzing the CXL Memory Simulator. However, we observed that the generated results were significantly better compared to the short prompt cases. We hypothesize that this improvement is due to the models' ability to comprehend the entire

TABLE VI
LLM Error Rate on Code Generation with Short Prompts

LLM Model	Refuse to Generate	Human Inspected Error
Straight Forward		
GPT-4 Turbo	0.00%	57.14%
LLaMA2-7B	28.57%	100.00%
LLaMA2-13B	28.57%	100.00%
LLaMA2-70B	28.57%	85.71%
StripedHyena	2.13%	85.71%
Mistral	0.00%	100.00%
Claude3-Sonnet	0.00%	71.43%
Claude3-Opus	0.00%	57.14%
ICL		
GPT-4 Turbo	0.00%	42.86%
LLaMA2-7B	42.86%	85.71%
LLaMA2-13B	42.86%	85.71%
LLaMA2-70B	42.86%	85.71%
StripedHyena	14.29%	85.71%
Mistral	0.00%	85.71%
Claude3-Sonnet	0.00%	57.14%
Claude3-Opus	0.00%	57.14%
CoT		
GPT-4 Turbo	0.00%	57.14%
LLaMA2-7B	28.57%	85.71%
LLaMA2-13B	28.57%	100.00%
LLaMA2-70B	28.57%	85.71%
StripedHyena	14.29%	85.71%
Mistral	0.00%	100.00%
Claude3-Sonnet	0.00%	71.43%
Claude3-Opus	0.00%	42.86%

[8]https://github.com/kwang927/LLM_Memory_System_Transition

architecture, including how the function is called elsewhere and the functionality of files without explicit comments.

B. Code Generation Results

In contrast to the Code Analysis Task, both ICL and CoT can be employed when designing prompts for the Code Generation Task. However, even in the best cases, compiling the generated program and running it for metrics is not feasible since LLMs always have small mistakes. Consequently, comparisons between LLMs and prompting strategies are based on: a) whether the LLM can generate code; and b) whether the generated code is similar to the template (i.e., using the same interface, calling existing functions, etc.).

Table VI presents the error rates of the Code Generation Task using short prompts. "Refuse to Generate" follows the same definition as mentioned in Section VI-A. "Human Inspected Error" is based on the error rate in the generated files. Generated files are labeled as errors if any of the following cases occur more than three times: a) the generated code uses functions that have not been defined; b) the function name in the generated file is completely different from the original version; c) the generation is incomplete. We manually annotated the 21 files generated by the LLMs and reported the result in the table.

As with Code Analysis, LLaMA2 models were observed to refuse to generate code for certain patterns of prompts and are more likely to refuse to generate code when using ICL prompts.

The results for ICL+CoT in the short prompt cases were not included since most of the prompts exceed 4096 tokens, causing most models to fail to generate code. Nevertheless, it was observed that the error rate in the valid code generated using ICL and CoT is significantly lower than in the straightforward prompting cases.

Table IV demonstrates that long prompt cases lead to much better performance when generating code. Once again, the LLMs closest to fulfilling the goal are GPT-4 Turbo and Claude3-Opus, indicating that these are the current state-of-the-art LLMs.

VII. Conclusion

In this work, we present a system capable of understanding and analyzing existing repositories and transferring code into another form. Although the system may still generate minor bugs when programming, the results demonstrate promising signs that, in the near future, more advanced language models will be able to effectively handle complex computer systems and software development challenges.

Furthermore, we observe that with the recent increase in acceptable context length of LLMs, models capable of reading the entire content of a problem in a single prompt hold clear advantages when addressing intricate tasks such as the one proposed in this work. Concurrently, traditional prompting strategies, including in-context learning (ICL) and chain-of-thought (CoT), continue to perform well and enhance the capabilities of LLMs under complex task setting.

Acknowledgements

We want to express our deep appreciation to Dr. Jishen Zhao and TA Theodoros Michailidis for their valuable and helpful suggestions and guidance.

The work is supported by **MOOBO** (Merely On Our Budget, Obviously). We extend our gratitude to Anthropic and TogetherAI for generously providing $5 and $25 free credits, respectively, to our newly created accounts. Without their support, we might have had to save more money by skipping one or two more lunches.

For readers who are willing to cover our expenses for this work, please reach out to either one of the authors via email. We will be very willing to add your name to the Acknowledgements section with our highest respect. Thank you so much (in advance)!

REFERENCES

[1] Anthropic, "Introducing the next generation of claude," https://www.anthropic.com/news/claude-3-family, Mar 2024, available from: https://www.anthropic.com/news/claude-3-family.

[2] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," 2020.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[4] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[5] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[6] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.139

[7] GitHub, "Github copilot: Your ai pair programmer," https://github.com/features/copilot, 2023, accessed: 2023-03-11.

[8] A. Gu, K. Goel, and C. Re, "Efficiently modeling long sequences with structured state spaces," in *International Conference on Learning Representations*, 2021.

[9] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9118–9147.

[10] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[11] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 22 199–22 213. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/8bb0d291acd4acf06ef112099c16f326-Paper-Conference.pdf

[12] M.-A. Lachaux, B. Roziere, L. Chanussot, and G. Lample, "Unsupervised translation of programming languages," 2020.

[13] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, "Rethinking the role of demonstrations: What makes in-context learning work?" in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 11 048–11 064. [Online]. Available: https://aclanthology.org/2022.emnlp-main.759

[14] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph, "Gpt-4 technical report," 2024.

[15] M. Poli, J. Wang, S. Massaroli, J. Quesnelle, R. Carlow, E. Nguyen, and A. Thomas, "StripedHyena: Moving Beyond Transformers with Hybrid Signal Processing Models," 12 2023. [Online]. Available: https://github.com/togethercomputer/stripedhyena

[16] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023.

[17] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[19] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[20] Z. Wang, X. Liu, J. Yang, T. Michailidis, S. Swanson, and J. Zhao, "Characterizing and modeling non-volatile memory systems," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 496–508.

[21] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler *et al.*, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022.

[22] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 24 824–24 837.

[Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[23] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 24 824–24 837. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf

[24] Y. Yang, P. Safayenikoo, J. Ma, T. A. Khan, and A. Quinn, "Cxlmemsim: A pure software simulated cxl.mem for performance characterization," 2023.

[25] Z. Zhao, E. Wallace, S. Feng, D. Klein, and S. Singh, "Calibrate before use: Improving few-shot performance of language models," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 697–12 706.