RepoQA: Evaluating Long Context Code Understanding

Jiawei Liu [1 *] **Jia Le Tian** [1 *] **Vijay Daita** [1] **Yuxiang Wei** [1] **Yifeng Ding** [1]
Yuhan Katherine Wang [1] **Jun Yang** [1] **Lingming Zhang** [1]
: https://evalplus.github.io/repoqa.html

Abstract

Recent advances have been improving the context windows of Large Language Models (LLMs). To quantify the *real* long-context capabilities of LLMs, evaluators such as the popular *Needle in a Haystack* have been developed to test LLMs over a large chunk of raw texts. While effective, current evaluations overlook the insight of how LLMs work with long-context code, *i.e.,* repositories.

To this end, we initiate the RepoQA benchmark to evaluate LLMs on long-context code understanding. Traditional needle testers ask LLMs to directly retrieve the answer from the context without necessary deep understanding. In RepoQA, we built our initial task, namely *Searching Needle Function* (SNF), which exercises LLMs to search functions given their natural-language description, *i.e.,* LLMs cannot find the desired function if they cannot understand the description and code. RepoQA is *multilingual* and *comprehensive*: it includes 500 code search tasks gathered from 50 popular repositories across 5 modern programming languages. By evaluating 33 general and code-specific LLMs on RepoQA, we show *(i)* there is still a small gap between the best open and proprietary models; *(ii)* different models are good at different languages; and *(iii)* models may understand code better without comments.

1. Introduction

Recently, there has been a growing interest in applying Large Language Models (LLMs) to process long documents in challenging tasks. The long context capability of LLMs is especially vital for assisting or even automating (Jimenez et al., 2023) the development of real-world software projects built up with thousands or even millions of lines of code. For example, when working on a new repository, developers would want to ask questions about the repository, *e.g.,* how to locate a specific function over the massive lines of code, where long-context LLMs can be helpful.

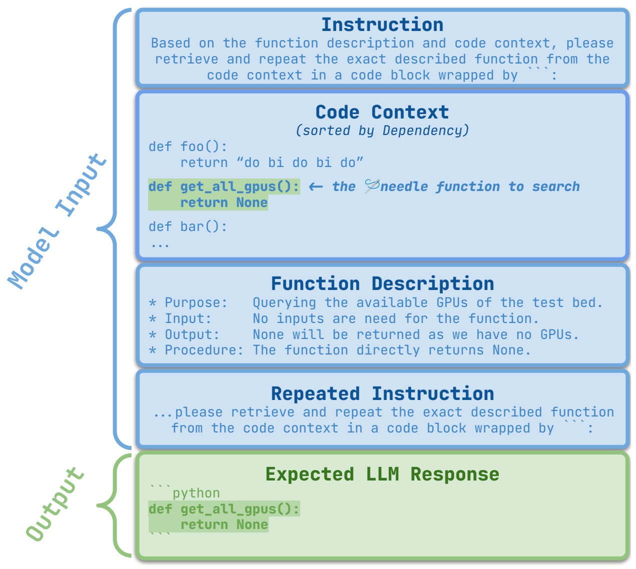
To quantify the long-context retrieval ability of LLMs, the *Needle in a Haystack* (gkamradt, 2023) (NIAH) benchmark was proposed. In this task, a random fact or statement (the "needle") is placed somewhere in the long context, such as a long story (the "haystack"). The model is then asked to answer a related question by retrieving this statement. The test is considered passed if the retrieved statement matches with the needle. Meanwhile, in the code domain, evaluators (Ding et al., 2023; Liu et al., 2023) have been introduced to benchmark long-context code generation instead of understanding.

While existing benchmarks for long-context understanding focus on general and synthetic use cases, to close the gap for code, we propose RepoQA, whose position is to exercise the *code understanding* ability of LLMs by creating tasks that can closely reflect *real-life* long-context uses. Specifically, inspired by code search (Paul & Prakash, 1994), in RepoQA we built our initial task called *Searching Needle Function* (SNF). Code search is a useful and real-life developer tool to search for related code (*e.g.,* functions) given a programmatic (GitHub, 2024) or natural-language (Husain et al., 2020) query. In SNF, we construct 500 code search tests from 50 repositories across 5 programming languages. Each test, as demonstrated by Figure 1, gives an LLM as the input: *(i)* instruction, *(ii)* a long context of code, *(iii)* the description of the desired function, and lastly *(iv)* a repetition of the instruction. By understanding the description and code, the model is expected to retrieve the desired function.

Below summarizes the contributions of RepoQA:

- **Dimension:** To our knowledge, RepoQA is the *first* benchmark for long-context code understanding.
- **Technique:** We propose an automatic pipeline to build evaluation sets for the *Searching Needle Function* task.
- **Artifact:** RepoQA is *multilingual* and *comprehensive*, covering 500 code search tasks gathered from 50 repositories across 5 modern programming languages.
- **Study:** Using RepoQA, we comprehensively evaluate 33 models and show interesting findings into the long-context abilities of current foundation models.

[*]Equal contribution. [1]University of Illinois Urbana-Champaign, USA. Correspondence to: Jiawei Liu <jiawei6@illinois.edu>.



Instruction

Based on the function description and code context, please retrieve and repeat the exact described function from the code context in a code block wrapped by ```:

Code Context
(sorted by Dependency)

```
def foo():
    return “do bi do bi do”
def get_all_gpus(): ← the needle function to search
    return None
def bar():
...
```

Function Description

```
* Purpose:   Querying the available GPUs of the test bed.
* Input:     No inputs are need for the function.
* Output:    None will be returned as we have no GPUs.
* Procedure: The function directly returns None.
```

Repeated Instruction

...please retrieve and repeat the exact described function from the code context in a code block wrapped by ```:

Expected LLM Response

~~~
```python
def get_all_gpus():
    return None
```
~~~

Figure 1. Exemplifying the *Searching Needle Function* task.

2. Related Work

As Large Language Models (LLMs) evolve, there is a clear trend toward improved handling of increasingly longer contexts. With recent models now supporting a 16k or larger context size by default (OpenAI, 2023; Gemini Team, 2024a; Jiang et al., 2024; Gemini Team, 2024b; Anthropic, 2024; Abdin et al., 2024), long-context benchmarks are becoming increasingly common. ZeroSCROLLS (Shaham et al., 2023) is a zero-shot benchmark featuring 10 task categories such as long-context question-answering and summarization. Concurrently, L-Eval (An et al., 2023) and LongBench (Bai et al., 2023) are proposed, where L-Eval comprises 20 sub-tasks with average input lengths from 4k to 60k and LongBench includes 21 datasets across 6 task categories in both English and Chinese. ∞Bench (Zhang et al., 2024) further extends the context window beyond 100k tokens for evaluating LLMs' capability in handling extremely lengthy documents. While these benchmarks mostly focus on realistic tasks, researchers have proposed synthetic long-context tasks to systematize the benchmark curation process. For example, *Needle in A Haystack* (gkamradt, 2023) (NIAH) involves hiding a fact (the ”needle”) in a long document (the “haystack”) and asking the model to retrieve this fact given a related question. RULER (Hsieh et al., 2024) expands upon the vanilla NIAH by providing four task categories with 13 representative tasks for long-context evaluation.

In the code domain, RepoBench (Liu et al., 2023) and CrossCodeEval (Ding et al., 2023) assess the ability of LLMs to perform repo-level code completion. While the two benchmarks require LLMs to process cross-file code context, the number of input tokens is still limited. SWE-BENCH (Jimenez et al., 2023) consists of 2,294 real-world software engineering problems, necessitating the models to do complex reasoning with long context. However, it is typically used to evaluate LLM-based code agents while being too complex for model-only evaluation. For example, GPT-4 only has a 1.31% pass rate using retrieval. Our work, RepoQA, fills the missing piece as the first benchmark for evaluating the core code understanding ability of LLMs over a very long context.

3. RepoQA

In this section, we introduce the design of RepoQA, which includes two main phases: *(i) data curation*: how to create long-context tests for the SNF task from repositories; and *(ii) model evaluation*: how to evaluate LLMs over these tests.

For data curation, we consider 50 repositories from 5 popular programming languages over various coding topics. For each repository, 10 evenly distributed needle functions are selected as the retrieval target and we prompt GPT-4 to annotate them with a natural-language description. During evaluation, we give the LLM under evaluation the corresponding code context and the function description and ask the LLM to repeat the corresponding function via an instruction. By comparing the model retrieved function against all function candidates within the context, the test is passed if the output is closest to the target function over a certain threshold of similarity.

3.1. Dataset Curation

Repository preparation. The input to the data curation pipeline of RepoQA is code repositories. Specifically, we select high-quality and popular GitHub repositories featuring various programming languages and application domains. Language-wise, we consider Python, C++, Java, TypeScript, and Rust, for their popularity and distinct positions in software engineering. For each language, we carefully selected 10 repositories, by mainly considering the following factors: *(i) topic diversity:* repositories spanning different topics (*e.g.,* “web” and “database”); *(ii) quality:* packaged repositories equipped with unit tests or CI/CD pipelines; and *(iii) popularity:* repositories with at least 100 GitHub stars.

For each selected repository, we define a root directory of the main package in the repository as the *entry directory*. As such, we only perform dataset creation over source files under this entry directory and ignore other less relevant files. Meanwhile, we perform dependency analysis to annotate these source files with their corresponding file dependencies, which will be used in later steps.

Needle function selection. The objective of SNF is to retrieve a desired function, which is termed *needle functions* in our work, inspired by the pioneering NIAH task. For each repository, we automatically select 10 needle functions evenly distributed over the repository. First, we construct a big source file by concatenating all source files under the entry directory in a topologically sorted order. We then split the big file into k ($k = 64$) evenly sized chunks, for each of which we collect the first function which has a unique function name and a reasonable function body length (*i.e.,* < 2000 bytes). Lastly, we randomly sample 10 out of the maximum k functions from the k chunks and use them as the needle functions for the repository.

Function description annotation. In SNF, we give the tested LLM a function description and ask it to retrieve the corresponding function. For evaluation accuracy, it is important to make the descriptions explicit and unique as vague and general descriptions can lead to multiple functions being reasonably mapped. Therefore, in our design, we decompose the description into four sections: function purpose, input description, output description, and general procedures, which are exemplified in the "Function Description" section in Figure 1.

We obtain these descriptions by using GPT-4-Turbo as the annotator with a prompt in Listing 1. Besides asking the annotator to output the four sections, the prompt asks the model to not reveal the function name and variable names of the needle function to avoid degrading to a simple keyword-matching problem. Meanwhile, the prompt also asks the model to specialize the description to differentiate it from other functions.

3.2. Model Evaluation

Context construction. We use each needle function to create a long-context test by constructing a task prompt shown in Figure 1. As the input to the tested model, it includes four components:

1. **Instruction:** A brief instruction clarifying the task.
2. **Code context:** A long sequence of N tokens (by the CodeLlama (Rozière et al., 2023) tokenizer) of code context including the needle function and other functions for obfuscation. To test if the model can retrieve the needle function at various context depths, within each repository, we plant the 10 needle functions over evenly paced depths, *e.g.,* depths of $10\%, 20\%, \cdots, 100\%$. The surrounding context of each needle function is derived from the repository code arranged in topological order.
3. **Function description:** A description of the needle function to search, as described in Section 3.1.
4. **Repeated instruction:** Finally, we close the input prompt by repeating the instruction, as prior work (Agrawal et al., 2024) shows that it can help remind the model of the task.

Score computation. Given the input prompt, the tested model produces outputs, based on which we decide if the model really finds the needle function successfully. By aggregating the success rate of retrieving needle functions across all tests, we obtain a final accuracy score from 0 to 100 (%) to represent the long-context code retrieval ability of the model.

We take a few steps to decide if a retrieval is successful from the model output. First, we perform post-processing to extract the first code block whose code is syntactically correct checked by tree-sitter. Next, given all possible functions within the code context $F = \{f_1, f_2, \cdots, f_n\}$, the needle function $\hat{f} \in F$, and the model produced function f_o, a successful retrieval is determined by satisfying two conditions:

(i) the model produced function f_o should be the most similar, defined by BLEU score with a smoothing function (Chen & Cherry, 2014), to the needle function $\hat{f}$ compared to other functions in F:

$$\hat{f} = \text{argmin}\{BLEU(f_i, f_o); f_i \in F\}$$

(ii) the similarity between the model produced function f_o and the needle function $\hat{f}$ should be no smaller than a user-given threshold (by default 0.8 in our work), to make sure f_o look close enough to $\hat{f}$:

$$BLEU(\hat{f}, f_o) > thresh$$

4. Evaluation

Experimental setup. We tested 33 major models on the 500 tasks in RepoQA and reported their retrieval accuracy on individual languages. Specifically, we let the token size of the code context be 16k (in Figure 1) and require a minimal similarity threshold of 0.8 between the model-generated function and the needle function (Section 3.2). We choose 16k as the code context size as Table 2 later shows that most major models can meet these criteria. Notably, using 16k code context can require a larger context length, as both other parts in the prompt and output (Figure 1) require additional token consumption. Therefore, for models with only 8k and 16k context sizes, we consider two training-free methods to unlock their context limit: *(i)* dynamic RoPE scaling (bloc97, 2023); or *(ii)* directly overwriting the maximum length. Of these, we report the best results.

Overall results. Table 1 lists the results of top-performing models, while we defer a more comprehensive table in Table 2 due to space limits. Overall we can see that the best-tier models (*i.e.,* with over 90% accuracy) are mostly proprietary models. The best-tier open-source models on this task are DeepSeek-V2-Chat and Llama-3-70B-Instruct which achieve over 80% accuracy, slightly outperforming proprietary models such as Claude-3-haiku and GPT-4-Turbo. Meanwhile, within the same model family, most larger models perform better than smaller ones, except for CodeLlama-13B-Instruct surpasses the 33B version. Lastly, the Llama-3-Instruct model family turns out to be secretly long-context models, achieving outstanding performance by simply extending the context using dynamic RoPE scaling.

#	Model	Ctx Size	`.py`	`.cpp`	`.rs`	`.java`	`.ts`	Avg. (CF)
1	gemini-1.5-pro-latest	1000k	91	81	91	94	**96**	90.6 (90.2)
	gpt-4o-2024-05-13	128k	95	80	85	96	**97**	90.6 (93.2)
3	gemini-1.5-flash-latest	1000k	93	79	87	94	**97**	90.0 (54.2)
4	DeepSeek-V2-Chat	128k	90	76	77	**91**	83	83.4 (84.4)
5	Meta-Llama-3-70B-Instruct*	8k	83	70	81	86	**91**	82.2 (86.2)
6	c4ai-command-r-plus	128k	81	74	76	**84**	77	78.4 (79.2)
7	gpt-4-turbo-2024-04-09	128k	84	79	75	**89**	55	76.4 (92.6)
8	Mixtral-8x7B-Instruct-v0.1	32k	66	65	64	71	**74**	68.0 (71.4)
9	Mixtral-8x22B-Instruct-v0.1	64k	60	67	74	**83**	55	67.8 (79.4)
10	Qwen1.5-72B-Chat	32k	62	60	68	**75**	70	67.0 (68.0)
11	Phi-3-medium-128k-instruct	128k	56	54	62	69	**74**	63.2 (71.2)
12	Mistral-7B-Instruct-v0.3	32k	61	56	51	61	**80**	62.0 (69.8)
13	Meta-Llama-3-8B-Instruct*	8k	54	48	51	53	**62**	53.6 (56.2)
14	deepseek-coder-33b-instruct	16k	59	44	23	53	**63**	48.4 (75.4)
15	Mistral-7B-Instruct-v0.2	32k	38	50	44	45	**60**	47.4 (54.0)

Table 1. Retrieval accuracy (%) of representative models with a matching threshold of 0.8. "CF" stands for the result differences when using the *comment-free* mode. "*" Denotes models evaluated using *dynamic ROPE scaling*. The full list can be found in Table 2.

Impact of natural comments. In Table 1 we also study the impact of natural-language code comments on retrieval accuracy. Intuitively, if the comment includes similar explanations and keywords to the query (*i.e.,* function description), it should be easier to retrieve compared to no comments. Surprisingly, Table 1 shows that the performance actually improves for models when comments are removed, except for the Gemini models. This potentially indicates LLMs can perform code understanding similarly or even better without comment assistance. Specifically, we perform comment removal using two steps to make comparison less sensitive to positional bias: (i) remove all comments in the code context, and (ii) use synthetic comment (*e.g.,* `# {LINE_NUMBER}` in Python) to pad the context to 16k while aligning the relative position with the default version. By debugging failure cases by Gemini-1.5-Flash, we see that the model often forgets the original task and simply continues counting the line numbers in the comment-free mode.

Impact of programming languages. By looking at the score distribution across different programming languages, we can see that most models are doing best at Java and TypeScript, followed by Python, C++, and then Rust, with some small-model outliers such as CodeLlama-7B-Instruct, Phi-3-mini-128k-instruct, and DeepSeek-Coder-6.7b-instruct doing best on C++. Interestingly, the difficulty of programming languages on RepoQA might be related to the amount of code corpus in their training set. For example, in the deduplicated version of Stack v2 dataset (Lozhkov et al., 2024), JavaScript/TypeScript and Java have the leading amount of corpus.

5. Conclusion and Future Work

Conclusion. We present RepoQA, a benchmark evaluating the long context understanding of LLMs. The benchmark currently contains the *Searching Needle Function* (SNF) task which asks the LLMs to fetch a given function given its description in natural language. By evaluating 33 models on this benchmark, we found that proprietary models still outperform the best open-source models, performance across languages differs depending on the model and removing comments may help with model understanding.

Future work. We hope to expand the scope of RepoQA in two ways, *(i) expanding Searching Needle Function* and *(ii) constructing more complex tasks*.

To expand the SNF task, we plan to include more programming languages and more models. We also hope to expand RepoQA through more complex scenarios, such as multi-hop retrieval and reasoning (Maharana et al., 2024).

Acknowledgement

We thank Google Deepmind and Weights & Biases for providing part of the compute. We especially thank Songrun Xie for his discussion and help in the early stages of RepoQA.

References

Abdin, M., Jacobs, S. A., Awan, A. A., Aneja, J., Awadallah, A., Awadalla, H., Bach, N., Bahree, A., Bakhtiari, A., Bao, J., Behl, H., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, Q., Cai, M., Mendes, C. C. T., Chen, W., Chaudhary, V., Chen, D., Chen, D., Chen, Y.-C., Chen, Y.-L., Chopra, P., Dai, X., Giorno, A. D., de Rosa, G., Dixon, M., Eldan, R., Fragoso, V., Iter, D., Gao, M., Gao, M., Gao, J., Garg, A., Goswami, A., Gunasekar, S., Haider, E., Hao, J., Hewett, R. J., Huynh, J., Javaheripi, M., Jin, X., Kauffmann, P., Karampatziakis, N., Kim, D., Khademi, M., Kurilenko, L., Lee, J. R., Lee, Y. T., Li, Y., Li, Y., Liang, C., Liden, L., Liu, C., Liu, M., Liu, W., Lin, E., Lin, Z., Luo, C., Madan, P., Mazzola, M., Mitra, A., Modi, H., Nguyen, A., Norick, B., Patra, B., Perez-Becker, D., Portet, T., Pryzant, R., Qin, H., Radmilac, M., Rosset, C., Roy, S., Ruwase, O., Saarikivi, O., Saied, A., Salim, A., Santacroce, M., Shah, S., Shang, N., Sharma, H., Shukla, S., Song, X., Tanaka, M., Tupini, A., Wang, X., Wang, L., Wang, C., Wang, Y., Ward, R., Wang, G., Witte, P., Wu, H., Wyatt, M., Xiao, B., Xu, C., Xu, J., Xu, W., Yadav, S., Yang, F., Yang, J., Yang, Z., Yang, Y., Yu, D., Yuan, L., Zhang, C., Zhang, C., Zhang, J., Zhang, L. L., Zhang, Y., Zhang, Y., Zhang, Y., and Zhou, X. Phi-3 technical report: A highly capable language model locally on your phone, 2024.

Agrawal, D., Gao, S., and Gajek, M. Can't remember details in long documents? you need some r&r. *arXiv preprint arXiv:2403.05004*, 2024.

An, C., Gong, S., Zhong, M., Zhao, X., Li, M., Zhang, J., Kong, L., and Qiu, X. L-eval: Instituting standardized evaluation for long context language models, 2023.

Anthropic. Introducing the next generation of claude anthropic. https://www.anthropic.com/news/claude-3-family, 2024.

Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., Dong, Y., Tang, J., and Li, J. Longbench: A bilingual, multitask benchmark for long context understanding, 2023.

bloc97. Ntk-aware scaled rope allows llama models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation. https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have/, 2023.

Chen, B. and Cherry, C. A systematic comparison of smoothing techniques for sentence-level BLEU. In Bojar, O., Buck, C., Federmann, C., Haddow, B., Koehn, P., Monz, C., Post, M., and Specia, L. (eds.), *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pp. 362–367, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3346. URL https://aclanthology.org/W14-3346.

Ding, Y., Wang, Z., Ahmad, W. U., Ding, H., Tan, M., Jain, N., Ramanathan, M. K., Nallapati, R., Bhatia, P., Roth, D., and Xiang, B. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=wgDcbBMSfh.

Gemini Team. Gemini: A family of highly capable multimodal models, 2024a.

Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024b.

GitHub. Github code search. https://github.com/features/code-search, 2024.

gkamradt. Llmtest needle in a haystack – pressure testing llms. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023.

Hsieh, C.-P., Sun, S., Kriman, S., Acharya, S., Rekesh, D., Jia, F., Zhang, Y., and Ginsburg, B. Ruler: What's the real context size of your long-context language models?, 2024.

Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., and Brockschmidt, M. Codesearchnet challenge: Evaluating the state of semantic code search, 2020.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M.-A., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mixtral of experts, 2024.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. Swe-bench: Can language models resolve real-world github issues?, 2023.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Liu, T., Xu, C., and McAuley, J. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023.

Lozhkov, A., Li, R., Allal, L. B., Cassano, F., Lamy-Poirier, J., Tazi, N., Tang, A., Pykhtar, D., Liu, J., Wei, Y., et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

Maharana, A., Lee, D.-H., Tulyakov, S., Bansal, M., Barbieri, F., and Fang, Y. Evaluating very long-term conversational memory of llm agents., 2024.

OpenAI. Gpt-4 technical report, 2023.

Paul, S. and Prakash, A. A framework for source code search using program patterns. *IEEE Transactions on Software Engineering*, 20(6):463–475, 1994.

Rozière, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., Kozhevnikov, A., Evtimov, I., Bitton, J., Bhatt, M., Ferrer, C. C., Grattafiori, A., Xiong, W., Défossez, A., Copet, J., Azhar, F., Touvron, H., Martin, L., Usunier, N., Scialom, T., and Synnaeve, G. Code llama: Open foundation models for code, 2023.

Shaham, U., Ivgi, M., Efrat, A., Berant, J., and Levy, O. ZeroSCROLLS: A zero-shot benchmark for long text understanding. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 7977–7989, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.536. URL https://aclanthology.org/2023.findings-emnlp.536.

Zhang, X., Chen, Y., Hu, S., Xu, Z., Chen, J., Hao, M. K., Han, X., Thai, Z. L., Wang, S., Liu, Z., and Sun, M. ∞bench: Extending long context evaluation beyond 100k tokens, 2024.

Appendix

We include additional tables and figures in our appendix.

```
def make_prompt(fn_name: str, code_ctx: str):
    instruction = f'Can
     you **briefly** describe the purpose, input, output, and procedure of "{fn_name}"?'
    return f"""\
{instruction}

```
{code_ctx}
```

{instruction}

Please follow the format to complete the skeleton below:

---
1. **Purpose**: ...
2. **Input**: ...
3. **Output**: ...
4. **Procedure**: ...

{instruction}

Notes:
1. DO NOT reveal function names ({fn_name}) and variable names
2. Customize the description to differentiate it from other functions
"""
```

Listing 1. Needle function annotation prompt for GPT-4-Turbo

#	**Model**	**Ctx Size**	Python	C++	Rust	Java	TypeScript	**Average**
1	claude-3-opus-20240229	200k	93	83	88	**95**	94	90.6
	gemini-1.5-pro-latest	1000k	91	81	91	94	**96**	90.6
	gpt-4o-2024-05-13	128k	95	80	85	96	**97**	90.6
4	gemini-1.5-flash-latest	1000k	93	79	87	94	**97**	90.0
5	claude-3-sonnet-20240229	200k	88	81	85	**92**	91	87.4
6	DeepSeek-V2-Chat	128k	90	76	77	**91**	83	83.4
7	Meta-Llama-3-70B-Instruct*	8k	83	70	81	86	**91**	82.2
8	claude-3-haiku-20240307	200k	80	75	74	**90**	90	81.8
9	c4ai-command-r-plus	128k	81	74	76	**84**	77	78.4
10	gpt-4-turbo-2024-04-09	128k	84	79	75	**89**	55	76.4
11	Mixtral-8x7B-Instruct-v0.1	32k	66	65	64	71	**74**	68.0
12	Mixtral-8x22B-Instruct-v0.1	64k	60	67	74	**83**	55	67.8
13	Qwen1.5-72B-Chat	32k	62	60	68	**75**	70	67.0
14	Phi-3-medium-128k-instruct	128k	56	54	62	69	**74**	63.2
15	CodeQwen1.5-7B-Chat	64k	69	47	56	**74**	67	62.8
16	Mistral-7B-Instruct-v0.3	32k	61	56	51	61	**80**	62.0
17	gpt-3.5-turbo-0125	16k	43	65	60	**76**	57	60.4
18	Meta-Llama-3-8B-Instruct*	8k	54	48	51	53	**62**	53.6
19	deepseek-coder-33b-instruct	16k	59	44	23	53	**63**	48.4
20	Mistral-7B-Instruct-v0.2	32k	38	50	44	45	**60**	47.4
21	CodeLlama-13b-Instruct-hf*	16k	45	30	31	50	**56**	42.6
22	CodeLlama-34b-Instruct-hf*	16k	41	31	**53**	40	43	41.6
	DeepSeek-V2-Lite-Chat	32k	39	37	45	41	**46**	41.6
24	Phi-3-small-128k-instruct	128k	25	48	30	46	**49**	39.6
25	Qwen1.5-32B-Chat	32k	36	28	25	32	**48**	33.8
26	CodeLlama-7b-Instruct-hf*	16k	20	**41**	22	25	33	28.2
27	Qwen1.5-14B-Chat	32k	4	30	26	**36**	34	26.0
28	Magicoder-S-DS-6.7B	16k	27	21	7	25	**36**	23.2
29	Phi-3-mini-128k-instruct	128k	19	**25**	**25**	21	22	22.4
30	Mistral-7B-Instruct-v0.1	32k	10	9	10	11	**15**	11.0
31	deepseek-coder-6.7b-instruct	16k	11	**21**	2	3	16	10.6
32	Qwen1.5-7B-Chat	32k	1	**6**	2	2	3	2.8
33	codegemma-7b-it	8k	3	2	1	1	**4**	2.2

Table 2. Retrieval accuracy (%) of all evaluated models with a matching threshold of 0.8. "*" denotes models evaluated using *dynamic ROPE scaling*