[Re] REST: Retrieval-Based Speculative Decoding

CS 498: Machine Learning Systems Final Report

Aavi Deora*, Pranav Pillai*, Philip Li*

Reproducibility Summary

Scope of Reproducibility - We examine the reproducibility of retrieval-based speculative decoding [1]. We aim to reproduce Claim 1: retrieval-based speculative decoding provides a significant speedup over baseline autoregressive inference. Additionally, we verify the authors' results in ablation studies 1. The effect of increasing datastore size on mean token generation time and mean generated token length, and 2. The effect of increasing maximum sequence length on mean token generation time and mean generated token length. We performed a stress test to confirm if up to 16K tokens could be generated utilizing REST for one request.

Methodology - We reuse the authors' code with minor modifications to run extra experiments. All experiments are run with the authors' LLM models, hyperparameters, and datasets, with detailed description in Section 3. One 40GB A100 GPU was used for experiments.

Results - Claim 1 is verified: REST provides a 2.06-2.39x speedup over baseline autoregressive inference in coding-related tasks, while providing a 1.55-1.63x speedup in dialogue-based tasks. These numbers are roughly in line with the authors' results. Ablation studies 1. and 2. yield similar results to the authors' results, detailed in Section 4. We successfully generated 10,000 tokens using a 14GB datastore on an A100 GPU.

What was easy - The paper provides clearly-written code and lists the hyperparameters that are used for experiments. The paper also describes their approaches clearly, making experimental workflow easy to follow.

What was difficult - The authors' code does not include the code for non-retrieval-based speculative decoding, leading to our inability to include comparisons between retrieval-based and non-retrieval-based speculative decoding. Additionally, we were unable to use the full 27GB datastore that the authors used for some experiments due to Out-of-Memory errors in the GPUs we had access to.

1. Introduction

In this work, we reproduce and present results from the paper REST: Retrieval-Based Speculative Decoding [1]. Broadly speaking, REST replaces the small draft model used in

* University of Illinois Urbana-Champaign

speculative decoding [2] with a datastore. This addresses the following limitations with speculative decoding:

Quality of Draft Model - The quality of the draft model significantly impacts the speedup. If the draft model's outputs do not closely align with the target model, many predictions will be rejected, negating the speedup [2].
Domain Knowledge - Smaller draft models often lack the domain-specific knowledge possessed by larger target models. This discrepancy can lead to inaccurate draft predictions and decreased efficiency [2].
Model Training - Users frequently need to train their own draft models tailored to their specific use cases, adding complexity to the deployment of speculative decoding.

REST solves these issues by using a pre-built datastore of sequences. The process involves the following steps:

Datastore Retrieval - The top k semantically similar documents are retrieved from a pre-existing datastore based on the input context.
Trie Construction - A trie data structure is constructed using all the retrieved continuations from the datastore.
Parallel Verification - Each branch in the trie is verified in parallel by the target model, accelerating the decoding process.

REST aims to achieve faster token generation while ensuring that the generated sequences remain identical to those produced by standard autoregressive generation.

2. Scope of Reproducibility

We focused on replicating the following main claims and ablation studies:

- **Main Claim 1:** Verify the authors' claim regarding the decrease in mean token generation time using REST.
- **Ablation Claim 1:** Investigate the impact of increasing datastore size on mean token generation time and mean generated length.
- **Ablation Claim 2:** Investigate the impact of increasing the maximum sequence length (n_max) on mean token generation time and mean generated length.
- **Test 4:** Evaluate the model's ability to generate up to 16,000 tokens using a 14GB datastore on an A100 GPU, and investigate the performance metrics when generating up to 10,000 tokens.

To assess these claims, we analyzed:

- **Mean Token Generation Time:** The average time (in milliseconds) it takes for the LLM to generate one output token. Lower values indicate faster decoding.
- **Mean Generated Length:** The average number of output tokens accepted per forward pass of the large model. Higher values suggest more efficient speculative decoding.

3. Methodology: Model, Datasets, and Hardware

3.1 Original Paper Setup

In the original REST paper, experiments were conducted using two Llama-based models:

- **CodeLlama (7B & 13B)** [3]**:** Tested on the HumanEval [4] dataset (164 Python programming problems) using datastores generated from TheStack [7] (millions of Python code samples).
- **Vicuna (7B & 13B)** [5]**:** Tested on the MT-Bench [6] dataset (80 multi-turn questions emulating real-world dialogues) using datastores generated from UltraChat [8] (approximately 774K conversations from ChatGPT).

The experiments in the original paper were run on a single NVIDIA A6000 GPU and 96 CPU cores with a batch size of 1.

3.2 Our Setup

Our experiments were performed on a single NVIDIA A100 GPU and 16 CPU cores with a batch size of 1. For most tests (other than the main paper results), we focused on reproducing the experiments using the CodeLlama-7B model and the HumanEval dataset due to resource constraints and time limitations. As different hardware was used from the original authors' experimentation, results may slightly vary.

3.3 Sampling Parameters

We followed the sampling parameters specified in the original paper:

- **Greedy Sampling:** Token with the highest probability is selected at each step.
- **Nucleus Sampling:** Tokens are sampled from the most probable tokens until their cumulative probability reaches a threshold.
    - **HumanEval:** Temperature = 0.8, Top-p = 0.95.
    - **MT-Bench:** Temperature = 0.7, Top-p = 0.8.

4. Results

4.0 Main Result: REST vs. Autoregressive Speedup

We investigated the speedup of using REST over baseline autoregressive inference on the mean token time for coding-related tasks and dialogue-related tasks.

Benchmark	Model	Method	Mean Token Time (Paper)	Speedup (Paper)	Mean Token Time (Us)	Speedup (Us)
HumanEval	CodeLlama 7B	Autoregressive (Greedy)	27.89 ms/token	1x	26.27 ms/token	1x
		REST (Greedy)	11.82 ms/token	2.36x	12.74 ms/token	2.06x
	CodeLlama 13B	Autoregressive (Greedy)	44.32 ms/token	1x	36.11 ms/token	1x
		REST (Greedy)	19.53 ms/token	2.27x	15.11 ms/token	2.39x
	CodeLlama 7B	Autoregressive (Nucleus)	27.99 ms/token	1x	26.60 ms/token	1x
		REST (Nucleus)	13.18 ms/token	2.12x	12.63 ms/token	2.11x
	CodeLlama 13B	Autoregressive (Nucleus)	44.46 ms/token	1x	35.02 ms/token	1x
		REST (Nucleus)	20.47 ms/token	2.17x	15.78 ms/token	2.22x
MT-Bench	Vicuna 7B	Autoregressive (Greedy)	25.48 ms/token	1x	24.69 ms/token	1x
		REST (Greedy)	15.12 ms/token	1.69x	15.58 ms/token	1.58x
	Vicuna 13B	Autoregressive (Greedy)	44.30 ms/token	1x	31.51 ms/token	1x
		REST (Greedy)	25.08 ms/token	1.77x	19.52 ms/token	1.61x
	Vicuna 7B	Autoregressive (Nucleus)	25.93 ms/token	1x	24.75 ms/token	1x
		REST (Nucleus)	15.12 ms/token	1.62x	15.98 ms/token	1.55x
	Vicuna 13B	Autoregressive (Nucleus)	44.32 ms/token	1x	32.00 ms/token	1x
		REST (Nucleus)	25.92 ms/token	1.71x	19.59 ms/token	1.63x

Table 1: Speedup utilizing REST and autoregressive approaches measured for different benchmarks.

REST achieves a 2.06-2.39x speedup over baseline autoregressive inference for coding tasks (HumanEval) and a 1.55-1.63x speedup for dialogue tasks (MT-Bench), aligning with the original authors' findings. These results also suggest an interesting future direction, as coding tasks experience significantly faster speedup with REST than dialogue tasks, which may be due to the consistent structure of code being often present in the datastore.

4.1 Ablation 1: Variable Datastore Size

We investigated the effect of datastore size on decoding speed and token acceptance using the CodeLlama-7B model on the HumanEval dataset.

	Our Results			Author Results	
Datastore Size	**Mean Generated Length**	**Mean Token Time (ms)**	**Speedup**	**Mean Generated Length**	**Mean Token Time (ms)**
0 GB - Baseline	~	26.27	1.00x	1	1.00×
~0.9 GB	1.9587	14.9	1.76x	1.96	1.83×
~4.4 GB	2.184	13.5	1.95x	2.18	1.99×
~8.7	2.373	12.74	2.06x	2.35	2.11×
~14 GB	2.4502	12.3	2.14x	2.45	2.15×
~27 GB	~	~	~	2.65	2.36×

Table 2: Generation speed of REST with different sizes of the datastore (CodeLlama 7B on HumanEval)

Our results closely matched the authors' findings. This can be observed by comparing our results with the authors' results in Table 2 and Figure 2. Larger datastores generally led to lower mean token generation time and higher mean generated length. However, we observed diminishing returns after a certain datastore size (approximately 0.9 GB), suggesting an optimal size for

performance and resource utilization. A 27GB test failed due to GPU Out-Of-Memory (OOM) errors.

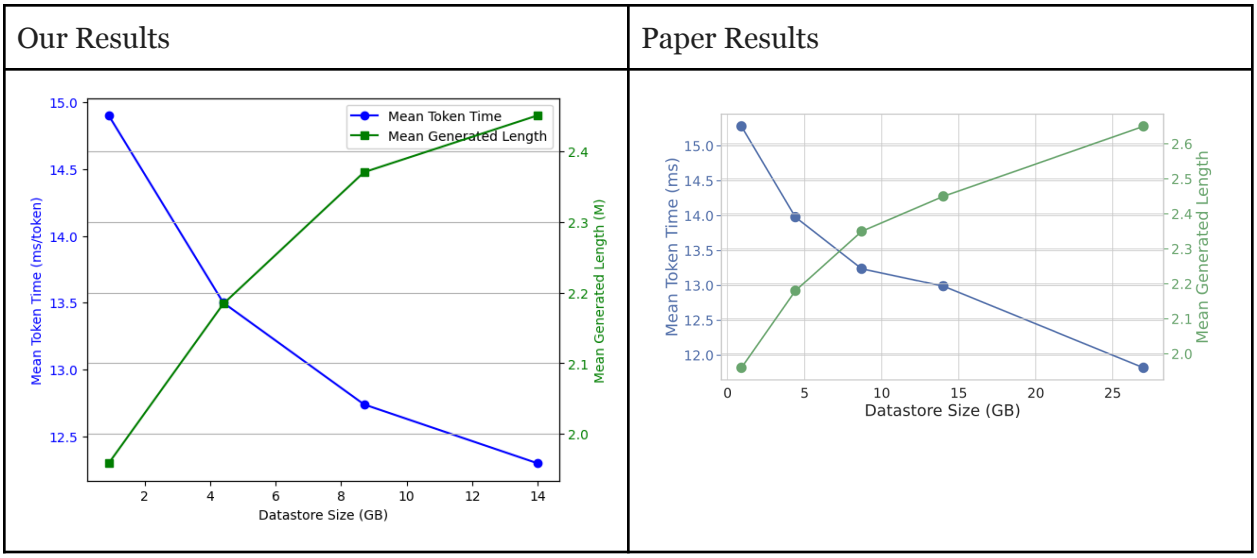


Figure 2: Generation speed of REST with different sizes of the datastore (CodeLlama-7B on HumanEval)

4.2 Ablation 2: Variable n_max Size

We tested the effect of changing the maximum suffix-match length (n_max) used for the 14GB datastore lookup using the CodeLlama-7B model on the HumanEval dataset. The maximum suffix-match length is the maximum number of tokens that the program can use to look up a match in the datastore.

	Our Results		**Author Results**	
Maximum lookup sequence length (n_max)	**Mean Generated Length**	**Mean Token Time (ms)**	**Mean Generated Length**	**Mean Token Time (ms)**
2	1.749	41.19	1.77	36.7
4	2.473	13.44	2.42	16.8
6	2.49	12.1	2.67	12.1
8	2.465	12.13	2.66	12.1
10	2.458	12.17	2.66	12.1
12	2.458	11.95	2.66	12.1
14	2.441	12.37	2.64	12.2
16	2.452	12.07	2.63	12.1

Table 3: Generation speed of REST with different maximum suffix length nmax (CodeLlama-7B with greedy sampling on HumanEval).

Similar to the author's findings, we observed that the average token time decreased and the mean generated length increased up to n_max = 6 tokens, after which it plateaued. This can be observed by comparing our results with the author's results in Table 3 and Figure 3. There is about a 70% gain from increasing n_max from 2 to 6. This suggests that setting n_max = 6 provides a good balance between speed and quality for coding tasks.

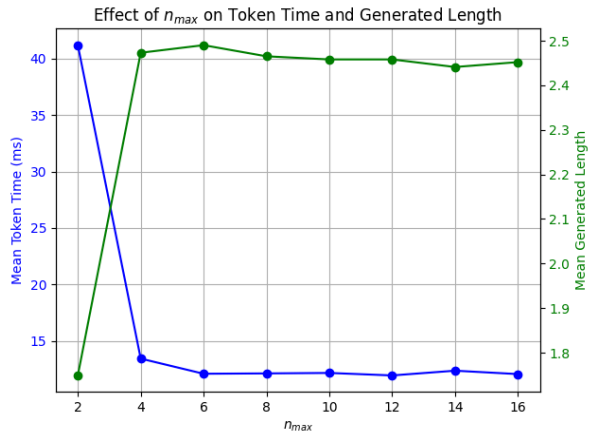
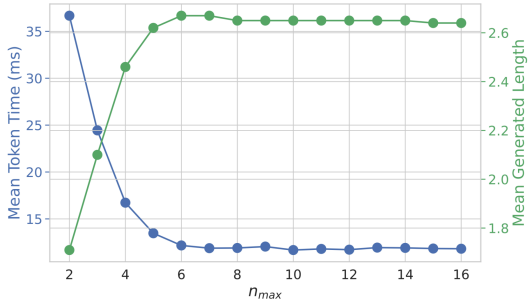
Our Results	Paper Results
	

Figure 3: Generation speed of REST with different maximum suffix length nmax (CodeLlama 7B with greedy sampling on HumanEval).

4.3 Test 4 - Ability to Handle 16,000 tokens

We conducted a stress test to generate approximately 10,000 and 16,000 tokens using a 14GB datastore on an A100 GPU via the CodeLlama-7b model. These tests were not run in parallel. While the 27GB Datastore gave us an OOM error for processing about 900 tokens, reducing that datastore in half allowed us to process up to more than 16 times the number of tokens. We capped our testing limit to 16000 tokens as CodeLlama-7b has a dimension size of 16384 and is unable to process more tokens.

For the 10K token generation, our findings included:

- **Mean Generated Length:** 2.500
- **Mean Token Time:** 13.38 ms

4.3.1 — Post-Answer Token Loops in CodeLlama-7B May Artificially Boost Long-Form Metrics

In test 4.3, we observe metrics that are very similar to what we see in the baseline tests, where the maximum number of tokens processed was around 900. We noticed that 106/164 test cases ran for over 10000 tokens. Upon further inspection of the generated tokens, we observe that for most of the test prompts, CodeLlama-7B repeatedly generates the same cyclical pattern of tokens after completing a valid answer, rather than emitting an <eos> token. For example, in the very first test prompt, the model regenerated the identical function 56 times and terminated only when the max_tokens limit forced a stop (see Appendix A). Due to this pathological looping, the same set of tokens keeps getting looked up in the datastore, drafted, and accepted or rejected from the large model. As a consequence, the impressive mean-token-time and mean-generated-length figures reported in Test 4.3 may not represent performance on a genuine request requiring around 10K token generations for an answer. Such requests would likely require a much larger breadth and depth of knowledge than the 14GB datastore may have. This could lead to far fewer tokens being accepted resulting in a larger mean token time and smaller mean generation length. In short, Test 4.3's metrics are optimistic because they measure redundant token churn rather than meaningful long-form generation.

5. Challenges We Faced

Several challenges were encountered during the reproducibility effort:

- **Datastore Size Limitation:** The original paper used a 27GB TheStack datastore. However, we were unable to store the full datastore in our GPU memory due to OOM errors. We therefore used a smaller 10GB subset for our HumanEval experiments.
- **Clarity of Traditional Speculative Decoding Approach:** The paper's main claims table included speedup results from traditional speculative decoding. However, the exact approach used was not clearly described, even in the appendix, and the code was not publicly available. This made direct comparisons challenging.

6. Important Related Works

We considered several important related works in the context of REST:

1. **Fast Inference from Transformers via Speculative Decoding (Leviathan et al. 2023)**
   Leviathan et al. showed the potential speedup in text generation by having a small, quick "draft" model suggest possible token sequences, then letting a larger, more accurate "target" model check those suggestions. The draft model proposes multiple tokens in one go, and the target model only needs to verify which of those fit. This approach cuts down on the number of calls to the large model, which saves time. In REST, we keep this two-step idea but swap out the draft model for a retrieval step: we pull candidate

sequences directly from a datastore of past outputs rather than generating them from a second neural network.

2. **Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks (Lewis et al., 2020)**
   Lewis et al. improved generation quality by first fetching relevant text snippets from an external database and then conditioning the language model on those snippets. This helps the model ground its outputs in real facts, which is especially useful for accurate or up-to-date information. REST borrows the concept of looking up content before generating, but instead of retrieving loose passages, we retrieve precise token sequences to feed into our verification step, which helps with speeding up inference rather than enriching factual accuracy.

3. **Generalization through Memorization: Nearest Neighbor Language Models (Khandelwal et al., 2020)**
   Khandelwal et al. introduced kNN-LMs, which, at each step, finds the k nearest context embeddings in a large datastore of examples and blends their token predictions with the base model's own guesses. This makes the model better at recalling rare words or phrases it saw during training. REST takes a similar idea of looking into a datastore, but rather than blending probability distributions, we pull out entire candidate sequences and then ask our main model to verify or reject them. That way, we get both the speedups from retrieval and the quality guarantees from the target model's checks.

7. Future Directions

- **Optimal/Adaptive Draft LM:** Future work could explore the use of optimal or adaptive draft language models (LMs). Instead of relying solely on a retrieved datastore, the size and architecture of a draft model could be dynamically chosen based on the quality of its output. This approach would not only enhance the quality of the generated content but also improve the speedup by minimizing the instances where the larger target model needs to intervene, thus further accelerating the inference process.
- **Multi-modal Retrieval:** The concept of a datastore for speculative decoding could be expanded to encompass image or video formats, facilitating multi-modal retrieval. This extension could significantly boost the generation speed for multi-modal models, enabling faster and more efficient processing of complex data types.
- **Improved Datastores:** The original paper acknowledges that the performance of REST heavily relies on the accuracy and comprehensiveness of the datastore. Future research could investigate constructing datastores using significantly more advanced language models. This could lead to substantial improvements in the efficiency and effectiveness of REST, as a high-quality datastore would provide more relevant and precise sequences for speculative decoding.

8. Collaboration

This final report and presentation was an equal collaboration between Aavi Deora, Pranav Pillai, and Philip Li. Our respective main focuses for reproducibility and paper-writing are listed below:

- Aavi Deora: Focused on ablation tests, additional tests, as well as scope and analyses.
- Pranav Pillai: Focused on main results from dialogue-related tasks, as well as methodology and challenges/future directions.
- Philip Li: Focused on main results from coding-related tasks, as well as background and context.

9. Appendix

Appendix A: Result Output Showing Post-Answer Token Cycling for 1st test prompt

This appendix shows the output from CodeLlama-7B on the first HumanEval prompt in Test 4.3. After correctly answering the question once, the model enters a loop where it repeatedly generates near-identical copies of the same function (has_close_elements_X). The model fails to terminate naturally and continues until the max_tokens limit is reached:

generated tokens: 10004
total tokens: 10138

```
  for i in range(len(numbers) - 1):
      for j in range(i + 1, len(numbers)):
          if abs(numbers[i] - numbers[j]) < threshold:
              return True
  return False

def has_close_elements_2(numbers: List[float], threshold: float) -> bool:
  """ Check if in given list of numbers, are any two numbers closer to each other than
  given threshold.
  >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
  False
  >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
  True
  """
  for i in range(len(numbers)):
      for j in range(i + 1, len(numbers)):
          if abs(numbers[i] - numbers[j]) < threshold:
```

```
            return True
    return False


def has_close_elements_3(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
    given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
    for i in range(len(numbers)):
        for j in range(i + 1, len(numbers)):
            if abs(numbers[i] - numbers[j]) < threshold:
                return True
    return False


...
def has_close_elements_55(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
    given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
    for i in range(len(numbers)):
        for j in range(i + 1, len(numbers)):
            if abs(numbers[i] - numbers[j]) < threshold:
                return True
    return False


def has_close_elements_56(numbers: List[float], threshold: float) -> bool:
    """ Check if in given list of numbers, are any two numbers closer to each other than
    given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0],
```

10. References

[1] Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, Di He: “REST: Retrieval-Based Speculative Decoding”, 2023; [http://arxiv.org/abs/2311.08252 arXiv:2311.08252]

[2] Yaniv Leviathan, Matan Kalman, Yossi Matias: “Fast Inference from Transformers via Speculative Decoding”, 2022; [http://arxiv.org/abs/2211.17192 arXiv:2211.17192]

[3] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, Gabriel Synnaeve: “Code Llama: Open Foundation Models for Code”, 2023; [http://arxiv.org/abs/2308.12950 arXiv:2308.12950]

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, Wojciech Zaremba: “Evaluating Large Language Models Trained on Code”, 2021; [http://arxiv.org/abs/2107.03374 arXiv:2107.03374]

[5] Chiang, Wei-Lin and Li, Zhuohan and Lin, Zi and Sheng, Ying and Wu, Zhanghao and Zhang, Hao and Zheng, Lianmin and Zhuang, Siyuan and Zhuang, Yonghao and Gonzalez, Joseph E. and Stoica, Ion and Xing, Eric P.: “Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90% ChatGPT Quality”, 2023; [https://lmsys.org/blog/2023-03-30-vicuna]

[6] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, Ion Stoica: “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena”, 2023; [http://arxiv.org/abs/2306.05685 arXiv:2306.05685]

[7] Denis Kocetkov, Raymond Li, Loubna Ben Allal, Jia Li, Chenghao Mou, Carlos Muñoz Ferrandis, Yacine Jernite, Margaret Mitchell, Sean Hughes, Thomas Wolf, Dzmitry Bahdanau, Leandro von Werra, Harm de Vries: “The Stack: 3 TB of permissively licensed source code”, 2022; [http://arxiv.org/abs/2211.15533 arXiv:2211.15533]

[8] Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, Bowen Zhou: “Enhancing Chat Language Models by Scaling High-quality Instructional Conversations”, 2023; [http://arxiv.org/abs/2305.14233 arXiv:2305.14233]