CODEMMLU: A MULTI-TASK BENCHMARK FOR ASSESSING CODE UNDERSTANDING CAPABILITIES OF CODELLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advancements in Code Large Language Models (CodeLLMs) have predominantly focused on open-ended code generation tasks, often neglecting the critical aspect of code understanding and comprehension. To bridge this gap, we present CodeMMLU, a comprehensive multiple-choice question-answer benchmark designed to evaluate the depth of software and code understanding in LLMs. CodeMMLU includes over 10,000 questions sourced from diverse domains, encompassing tasks such as code analysis, defect detection, and software engineering principles across multiple programming languages. Unlike traditional benchmarks, CodeMMLU assesses models' ability to reason about code rather than merely generate it, providing deeper insights into their grasp of complex software concepts and systems. Our extensive evaluation reveals that even state-of-the-art models face significant challenges with CodeMMLU, highlighting deficiencies in comprehension beyond code generation. By underscoring the crucial relationship between code understanding and effective generation, CodeMMLU serves as a vital resource for advancing AI-assisted software development, ultimately aiming to create more reliable and capable coding assistants.

1 INTRODUCTION

In recent years, Code Large Language Models (CodeLLMs) have witnessed significant advancements (Wang et al., 2021; 2023b; Feng et al., 2020; Allal et al., 2023; Li et al., 2023; Lozhkov et al., 2024b; Guo et al., 2024; Pinnaparaju et al., 2024; Zheng et al., 2024c; Roziere et al., 2023; Nijkamp et al., 2022; Luo et al., 2023; Xu et al., 2022; Bui et al., 2023), demonstrating impressive performance across various benchmarks (Iyer et al., 2018; Lu et al., 2021; Chen et al., 2021; Austin et al., 2021; Lai et al., 2023; Hendrycks et al., 2021; da Silva Maldonado et al., 2017; Palomba et al., 2015). These sophisticated models, trained on extensive datasets of programming languages and coding patterns, have demonstrated impressive capabilities in generating, summarizing, debugging, and refactoring code. Despite the promising progress, existing benchmarks are often outdated and prone to data leakage (Matton et al., 2024), resulting in less rigorous evaluations. Moreover, in practice, CodeLLMs reveal various shortcomings, including bias and hallucination (Rahman & Kundu, 2024; Liu et al., 2024a), that have not been fully investigated due to the limitations of current benchmarks' ability.

Coding-related benchmarks primarily emphasize on code generation or completion (Iyer et al., 2018; Lu et al., 2021; Chen et al., 2021; Austin et al., 2021; Lai et al., 2023; Hendrycks et al., 2021; Ding et al., 2023; Zhuo et al., 2024), thereby restricting their capacity to assess the model's capabilities in handling a variety of tasks, including free-form and multiple-choice question answering (MCQA). Moreover, since code generation requires reliable evaluation based on test cases and executability, these benchmarks are typically limited in both quantity and diversity across different domains. Consequently, the measurement could be potentially biased and lack generalizations. Several efforts (Liu & Wan, 2021; Li et al., 2024) have been introduced to improve evaluation capabilities through free-form question answering and to expand domain diversity. However, these benchmarks frequently involve trade-offs that necessitate the use of less rigorous evaluation metrics, such as match-based metrics. As a result, the evaluations tend to lack robustness and reliability.

To address the aforementioned shortcomings, we present CodeMMLU - the first comprehensive MCQA benchmark specifically designed for evaluating code and software knowledge. We follow the scenario introduced in the well-known MMLU dataset (Hendrycks et al., 2020) from the natural language understanding domain offering a robust and simple-to-evaluate approach. CodeMMLU is specifically developed to overcome the limitations of current benchmarks in software field, providing a reliable evaluation for LLMs with the following key focuses:

- **Comprehensiveness:** To address the issue of bias in a limited proportion of evaluation data, we carefully curated CodeMMLU from a diverse range of high-quality sources. In summary, we obtain over 10,000 questions for evaluation.
- **Diversity in task, domain, and language:** CodeMMLU is designed to provide a comprehensive assessment of LLM's code understanding capabilities. To this end, we have diversified the tasks within the dataset, covering a spectrum from general software knowledge QA to real-world challenges like code generation, defect detection, and code repair. Additionally, the dataset encompasses a wild range of topic domains, including database management, APIs and frameworks, and data structures and algorithms, across more than 10 programming languages. This approach allows for a more fine-grained assessment of LLM capabilities, enabling us to observe the strengths and weaknesses of each model.

The dataset enables us to assess LLMs capabilities in coding and software tasks from a different commonsense perspective, extending beyond code generation and completion. As a result, we identify previously unexplored biases issue in CodeLLMs, align to the problem observed in natural language MCQA using LLMs as shown in prior work of Zheng et al. (2023). Furthermore, our experiments involving a large number of LLMs and various prompting techniques, such as few-shot and chain-of-thought, demonstrate noticeable patterns related to the impacts of model size, model family, and inconsistencies across different prompting strategies. Specifically, the closed-source model GPT-4 consistently achieves the highest average performance. Meanwhile, the Meta-Llama family demonstrates the greatest accuracy among the closed-source model families. Besides, Scaling law related to model size is partially observed within models of the same family but is not evident across different families. This suggests a substantial influence of pretraining datasets, methodologies, and model architectures. Additionally, the performance and advancements in prompting techniques illustrate a lack of improvement, implying concerns regarding hallucinations in the reasoning ability of each model. In summary, our contributions are outlined as follows:

1. We present CodeMMLU, the first MCQA benchmark for software and coding-related knowledge. Addressing the need for diverse evaluation scenarios in code as in the NLP field, CodeMMLU enables the ability to evaluate LLMs' alignment with human inference with MCQA in the software knowledge domain.
2. CodeMMLU provides a comprehensive evaluation of LLM capabilities, ensuring a substantial number of samples and diversity across tasks, domains, and languages. Hence, it enables a more nuanced understanding of an LLM's strengths and weaknesses, facilitating the development of models that are better aligned with the complexities and demands of the software domain.
3. To mitigate the bias issue in LLMs on MCQA to ensure a robust and reliable evaluation, we propose a rigorous evaluation scenario that requires the correct answer to be identified across all permutations of the choices. This increases the difficulty of our dataset, necessitating the use of more robust and sophisticated models.
4. Our experiments offer valuable insights into LLM performance, highlighting the impact of factors such as model size, model family, and prompting techniques used. This provides important information to the community on utilizing LLMs for specific tasks and domains.

2 Related work

Code Generation and Understanding evaluation To evaluate and advance the capabilities of Large Language Models (LLMs) in code-related tasks, numerous benchmark datasets have been developed, covering a wide range of programming challenges. Algorithm-style benchmarks include HumanEval Chen et al. (2021) and MBPP Austin et al. (2021), along with their extended versions

HumanEval+, Multi-PL and MBPP+ Liu et al. (2024b), which focus on standard algorithms. More challenging algorithmic tasks are represented by CodeContests Li et al. (2022) and LiveCodeBench Jain et al. (2024), which are based on competitive programming problems. DS-1000 Lai et al. (2023) challenges LLMs with specific data manipulation and analysis tasks, MathQA-Python Austin et al. (2021) focuses on mathematical problem-solving in Python. Repository-level benchmarks such as RepoBench Liu et al. (2023), RepoEval Zhang et al. (2023), and SWE-Bench Jimenez et al. (2023) present more complex, real-world scenarios in software development. Additionally, comprehensive benchmarks like XCodeEval Khan et al. (2023), CRUXEval Gu et al. (2024a), and CodeXGLUE Lu et al. (2021) evaluate LLMs across multiple dimensions of software development.

Programming Comprehension Modeling The community have actively research for a model of the programmer behavior or cognitive process of programmer since the early day of software development Storey (2005); Shneiderman & Mayer (1979); Xia et al. (2018). A cognitive model's target to describe the task of the cognitive structures and cognitive processes, which are involved programming knowledge, concepts, techniques when comprehension; and processes the programmer involved when building a problem solution. Shneiderman & Mayer (1979) introduce the multi-leveled of cognitive structures which consist of measuring the semantic and syntactic knowledge. The semantic contains programming concepts and technique (e.g. dynamic programming, recursion or sorting and merging method); where syntactic related to programming language grammatical (e.g. format of iteration, condition, library function). In other hands, to measuring the programmer comprehension in term of cognitive processes, Shneiderman & Mayer (1979) propose 5 core foundation programming tasks to any programmer behavior model which be accountable of, namely composition, comprehension, debugging, modification, and learning. This model architecture are design to answer 2 important question in programmer comprehension task, (1) measuring what kind of knowledge is available to programmer and (2) what kind of process does the programmer go through during the designing solution phase.

Table 1: Comparison between common code understanding benchmarks for LLMs in term of coverage five foundation tasks of programming comprehension model. † and ⋆ denote the benchmark with the free-flow generation and multiple-choice question answering format, respectively.

Benchmark	Programming Task					#Tasks	Data size
	Programming Knowledge	*Code Composition*	*Code Comprehension*	*Code Modification*	*Code Debugging*		
APPS† Hendrycks et al. (2021)		✓				1	5
MBPP† Austin et al. (2021)		✓				1	974
HumanEval† Chen et al. (2021)		✓				1	164
CRUXEval† Gu et al. (2024b)			✓			2	800
LiveCodeBench† Jain et al. (2024)		✓	✓		✓	4	-
CodeApex⋆† Fu et al. (2023)	✓	✓			✓	3	2.056
CodeMMLU⋆	✓	✓	✓	✓	✓	**6**	**19.912**

Multi-task question answering benchmarks Multiple Choice Questions (MCQs) have become a significant tool for evaluating the capabilities of Large Language Models (LLMs) across various domains. Recent trends in MCQ-based benchmarks are focused on testing advanced reasoning, domain-specific knowledge, and robustness in LLMs, particularly in light of their expanding capabilities Hendrycks et al. (2020); Lin et al. (2022); Zellers et al. (2019); Talmor et al. (2019). Since the LLMs are evolving rapidly, MCQs method benefits are not only lie in enable scale evaluating, but also provide a high-reliability result. Comparing with other open-ended judging method Chiang et al. (2024) which heavily depend on LLM judge or human annotation effort, MCQs method enhance the reliability by grounded the knowledge, problem context and defined possible available answers. Despite the handy of MCQs, recent studies has shown LLMs display a significant sensitivity to the order of answer options in MCQs Wang et al. (2023a); Robinson et al. (2023). Therefore raise a request for applying appropriate debias methods on the MCQs benchmark to address the LLM selection bias Zheng et al. (2024b); Pezeshkpour & Hruschka (2024)

3 MMCLU: Data Creation

The CodeMMLU benchmark is constructed to assess large language models' (LLMs) comprehension of programming tasks. We are inspired by programmer comprehension behavior models and integrate multi-leveled cognitive structures and processes to measure the understandability of LLMs on software

problems (Shneiderman & Mayer, 1979). CodeMMLU is divided into two primary categories: (i) knowledge-based test sets containing syntactic and semantic tasks, and (ii) real-world programming problems. The overall CodeMMLU structure, as presented in figure 1, includes distinct approaches for data collection, filtering, and validation in both test sets.

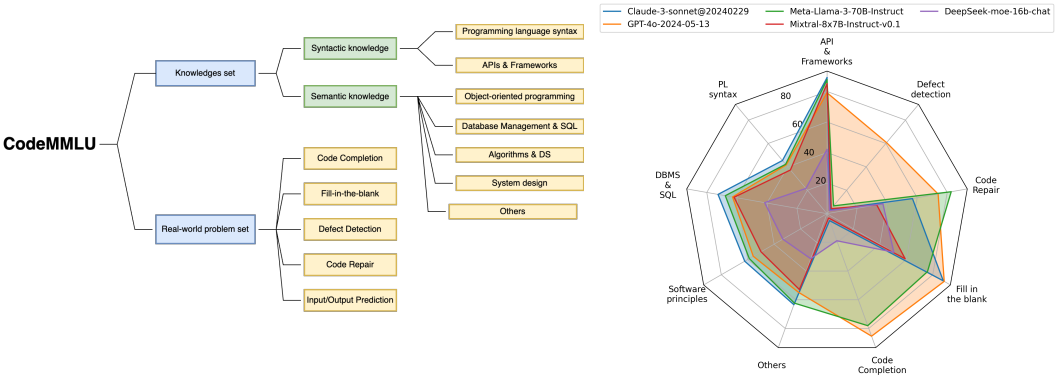

Figure 1: **Overview of the CodeMMLU Structure.** The CodeMMLU benchmark is divided into two distinct subsets, each designed to evaluate different aspects of LLMs' programming capability.

Figure 2: **Summary performance of LLMs on the CodeMMLU benchmark.** This radar chart presents the evaluation results (accuracy %) of different models across various CodeMMLU tasks.

3.1 KNOWLEDGE-BASED TASK CREATION

The knowledge-based subset covers wide range of topics, from high-level software principles concepts to low-level programming language grammars, targets to measuring the LLMs coding capability and comprehensibleness of programming concepts. We collected programming-related MCQs from high-quality platforms, namely GeeksforGeeks, W3Schools, and Sanfoundry GeeksforGeeks (2024); W3Schools (2024); SanFoundry (2024).

- **Syntactic set.** Focused on programming language grammar and structural correctness, such as condition statement, format of iteration, common library usage, etc.
- **Semantic set.** Targeted more abstract programming concepts, such as algorithm, data structures, object-oriented principles, etc.

Filtering Process. We manually categorize questions into subject based on the topic of collected data. These processed questions are then applied a deep-learning-based filtering models to auto-

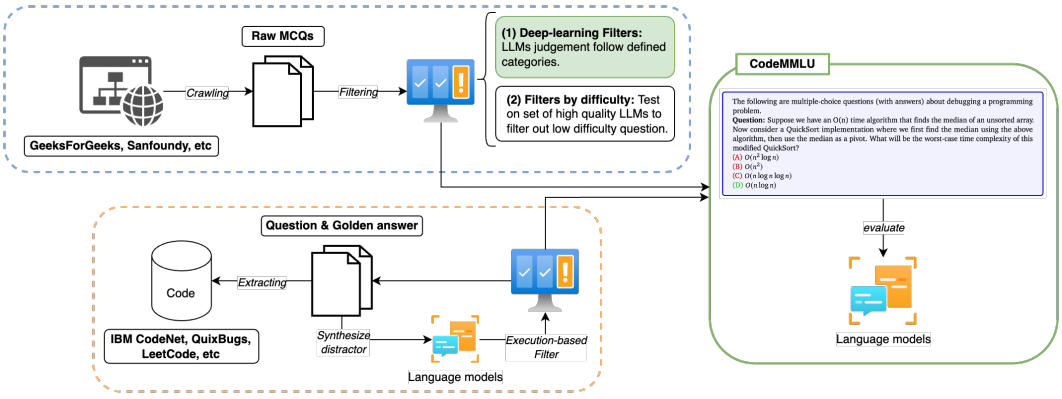

Figure 3: **Overview of CodeMMLU data creation pipeline.** The blue diagram describe the process of collecting raw multiple-choice questions (MCQs) from open source internet for a knowledge testset. Otherwise, the pipeline of real-world problem indicated in orange area.

Table 2: **Summary of CodeMMLU Subject Categories and Task Distribution.**

	Subject	Topic	Source	Testsize
Syntactic knowledge	API & Frameworks usage	Jquery, Django, Pandas, Numpy, Scipy, Azure, Git, AWS, svg, xml, Bootstrap, NodeJS, AngularJS, React, Vue.	W3Schools, Geeks4Geeks, Sanfoundry	740
	Programming language syntax	C, C#, C++, Java, Javascript, PHP, Python, R, Ruby, MatLab, HTML, CSS, TypeScript.		6,220
Semantic knowledge	DBMS & SQL	DBMS, MySQL, PostgreSQL, SQL.		393
	Software principles	Data structure & Algorithm, Object-oriented programming, Compiler design, Computer organization and Architecture, Software Development & Engineering, System Design.		3,246
	Others	Program accessibility, Computer networks, Computer science, Cybersecurity, Linux, Web technologies, AWS.		1,308
Real-world task	Code completion		HumanEval	163
	Fill in the blank		LeetCode	2,129
	Code repair		QuixBugs	76
	Defect detection		IBM CodeNet	6,006

matically discard low-quality or irrelevant questions. For example, we removed duplicate or trivial question which did not challenge the code comprehension capabilities of LLMs (Appendix A.2.1). The final knowledge-based subset comprises of a manual and deep-learning-based filters to ensure each question met the desired quality standards, which must be clear, the unambiguous, and challenging to evaluate both semantic and syntactic understanding of LLMs. The filtering process returns a knowledge-based subset contains approximately 6,000 syntactic questions and over 3,000 question focus on semantic, cover 52 topics classified into 5 main subjects (Table 2).

3.2 REAL-WORLD TASK CREATION

Composition, comprehension, debugging, and modification are the core capabilities that any cognitive model of programmer behavior must address. We includes five distinct MCQ programming tasks, designed to assess the foundational abilities outlined in the cognitive process model of programmer comprehension, namely: Code completion; Code repair; Defect Detection; and Fill in the blank.

Code Completion covers the composition aspect in programming behavior model and evaluates a model's ability by request to complete partially written code based on the provided requirement. We adapted the HumanEval Chen et al. (2021), originally designed for code generation, to and MCQ. From their 164 unique programming problems, we employed large language models (LLMs) to generate alternative, plausible but incorrect solutions, known as distractors. These alternatives were crafted to closely resemble plausible but incorrect solutions. All generated solutions, including both correct (migrate from HumanEval solutions) and incorrect options, were tested to ensure they are executable. Some of the incorrect solutions were designed to pass certain test cases but fail others. We target to add a layer of complexity, challenging the model to distinguish between correct and nearly-correct solutions based on semantic and syntactic understanding.

Code Repair is designed to evaluate model's debugging capability by requiring to identify and fix errors in provided code snippets. We create Code Repair upon the QuixBugs Lin et al. (2017), which originally designed for debugging algorithmic programs. We use a "diff" operation on buggy and corrected version in QuixBugs in Python and Java to identify the specific changes needed to fix the bugs, which later served as the correct solution. To create plausible distractors for the MCQ format, we targeted components are known to be frequently involved in bugs, namely return statement, loop condition, if/else/switch expressions, and guided a LLMs to generate alternative fixes based on these components. These alternatives were intentionally designed to mislead by suggesting changes that seem plausible but do not (fully) resolve the bug. Each distractor was verified for incorrectness, and all options were executable to ensure that the model needed a deep understanding of the code to correctly identify and apply the necessary fix.

Defect Detection aims to evaluate a model's ability to identify and understand defects within a code snippet, focusing on both logical and syntactical errors. We measure the comprehension and debugging capability of the LLMs by requiring it to predict the execution outcome of a given piece of code. The task include two sub-tasks: detecting any defection/flaw in provided code, and comprehend the output of certain test sample. The Defect Detection set derived from IBM CodeNet Puri et al. (2021), a large-scale, high-quality benchmark designed for algorithmic coding tasks. IBM Project CodeNet include over 14 million code samples submitted by users to solve a variety of algorithmic challenges in multiple programming languages. From the vast pool of user-submitted solutions, we focused on the Python and Java subsets and collected both accepted and buggy versions of code. To ensure a high-quality and representative dataset, we filtered out duplicate submissions and retained a diverse set of code samples. To create the Defect Detection task, we mixed the buggy versions with their correct counterparts, ensuring a balanced representation of various types of errors. For each code snippet, there will be a corresponding execution result (golden answer) and a random of three others distracting option, which could be one of several possible outcomes, including: (i) Compile Error, (ii) Time Limit Exceeded, (iii) Memory Limit Exceeded, (iv) Internal Error, (v) Runtime Error and (vi) The code not contain any issue.

Fill in the blank asks the model to fill in missing parts of a code snippet, given documentation and incomplete code snippet. We designed this task to assess a model's ability to comprehend code, not only because filling in the empty task, but models also be tested the understanding of both high-level programming concepts and low-level grammatical structures, since problem requires models to correctly identify and complete the incomplete parts of a program. We collected approximately 2,000 coding problem from LeetCode, covering solution from three wisely-use programming language (Python, Java, C++). From each problem's solution, we randomly selected key components to be blanked out. These components were chosen for their importance in the logic and flow of the program, for example loop conditions, expression statements, conditional statements.

To create plausible but incorrect options for the multiple-choice question (MCQ) format, we used large language models (LLMs) to generate alternative solutions for the blanked-out components. These distractors were designed to be contextually relevant but incorrect, adding complexity to the task. After generating these distractors, we executed them to verify their incorrectness, ensuring that they do not solve the problem as intended. To further enhance the difficulty of the task, we normalized all variable names and function names within the code snippets. This normalization process involved replacing original names with generic placeholders, such as 'var1', 'var2', 'func1', etc. This step reduces the reliance on specific naming conventions and forces the model to focus purely on the logic and structure of the code, rather than relying on context clues from variable or function names.

4 Experimental Results

4.1 Setup

Model selection. We evaluated 35 top-tier open-source models, covering a wide range of parameter sizes and architectures. The models were selected from 7 different families, with parameters ranging from 1 billion to over 70 billion. Each family included base, instructed, and chat versions: MetaLlama3.1/8B/70B Dubey et al. (2024), MetaLlama3/8B/70B AI@Meta (2024), CodeLlaMA/7B/13B/34B Rozière et al. (2024), DeepSeek-ai/6.7B/7B/33B Guo et al. (2024), MistralAI/8x7B Jiang et al. (2024), Qwen2/7B qwe (2024), CodeQwen1.5/7B Bai et al. (2023), Yi/6B/9B/34B AI et al. (2024), StarCoder2/7B/15B Lozhkov et al. (2024a). In addition to open-source models, we included 3 proprietary models from OpenAI and Claude to ensure comprehensive coverage of the state-of-the-art in language modeling: GPT-3.5/GPT-4 OpenAI et al. (2024), Claude-3-opus/Claude-3.5-sonnet The.

Answer extraction. CodeMMLU leverages the MCQ format for scalability and ease of evaluation. In order to maintain this advantage, we only apply simple regex methods to extract the selection answer (i.e. extract by directly answering or contains the pattern "answer is A|B|C|D"). The model response is required to be parsable; otherwise, it will be marked as unanswered.

Prompting. We employed various prompt strategies to test model performance across different scenarios, namely: Zeroshot, Fewshot, CoT, and CoT Fewshot.

All experiments were conducted on a cluster of 8 A100 GPUs.

4.2 KEY INSIGHTS

Table 3: **Summary performance of LLM family on CodeMMLU**. The evaluation results (accuracy %) of different language models across CodeMMLU task.

Family	Model name	Size (B)	Knowledge test		*Real-world tasks*	CodeMMLU
			Syntactic	*Semantic*		
		Closed-source models				
Anthropic	Claude-3-sonnet@20240229	-	67.22	**66.08**	38.26	53.97
OpenAI	GPT-4o-2024-05-13	-	60.41	57.82	77.18	**67**
	GPT-3.5-turbo-0613	-	61.68	84.88	58.52	51.7
		Open-source models				
Meta Llama	CodeLlama-34b-Instruct-hf	34	56.81	46.93	23.55	38.73
	Meta-Llama-3-70B	70	63.38	86.02	63.1	48.98
	Meta-Llama-3-70B-Instruct	70	64.9	**87.59**	67.68	**62.45**
	Meta-Llama-3.1-70B	70	64.09	87.02	65.65	37.56
	Meta-Llama-3.1-70B-Instruct	70	64.42	87.45	**69.21**	60
Mistral	Mistral-7B-Instruct-v0.3	7	54.42	51.25	31.85	43.33
	Mixtral-8x7B-Instruct-v0.1	46.7	61.17	85.02	61.83	42.96
	Codestral-22B-v0.1	22	60.34	82.17	58.52	47.6
Phi	Phi-3-medium-128k-instruct	14	58.54	54.56	37.89	48.03
	Phi-3-mini-128k-instruct	3.8	53.01	74.18	54.2	37.93
Qwen	Qwen2-57B-A14B-Instruct	57	61.34	57.48	30.48	46.34
	CodeQwen1.5-7B-Chat	7	49.66	67.62	46.06	49.82
Yi	Yi-1.5-34B-Chat	34	58.32	55.59	40.27	49.39
	Yi-1.5-9B-Chat	9	55.64	75.46	60.05	47.23
Deep Seek	DeepSeek-coder-7b-instruct-v1.5	7	56.67	47.9	28.46	41.21
	DeepSeek-coder-33b-instruct	33	53.65	74.89	52.16	36.6
	DeepSeek-moe-16b-chat	16.4	31.74	41.94	41.48	31.01
	DeepSeek-Coder-V2-Lite-Instruct	16	59.91	81.74	64.38	46.51
InternLM	InternLM2-5-20b-chat	20	57.85	55.51	30.44	44.89
StarCoder2	StarCoder2-15b-instruct-v0.1	15	56.58	49.07	42.79	47.94

The CodeMMLU benchmark revealed significant performance differences across models, as follow in Tab. 3. The GPT-4o from OpenAI outperformed all models on CodeMMLU, reflecting its quality across diverse tasks (Fig. 2). On the other hand, despite not being the latest model, the Meta-Llama-3-70B instructed version achieves the highest score among open-source models from 8 families.

The Fig. 4 indicates the capability of measuring LLMs coding knowledge and skill in a wide range of subjects of CodeMMLU. Our benchmark provides clear, distinct rankings that bring a higher hierarchy of models than other code generation benchmarks. Interestingly, the result are not following the scaling laws Kaplan et al. (2020), where models with larger parameter sizes outperform the smaller. This shows the effect of data quality in the pretraining LLMs process to LLMs ability, where recently released models are likely to achieve comparable performance with the large size model from previous version. The importance of instruction tuning in improving the model's performance in complex tasks also indicated in CodeMMLU, while the prompt technique (i.e. Chain-of-thought Wei et al. (2023)) shows ineffective in boosting the comprehension of the models (Fig. 8. Models with instruction tuning achieve substantial higher than its non-instruct counterpart, like DeepSeek-Coder-33b is superior then its base model by approximately 29%.

Correlation Between Software Knowledge and Real-World Performance Our experiments found a strong correlation between performance on knowledge-based tasks and real-world coding tasks. The correlation between model ranks on the knowledge test set and real-world problems, by TODO: x in Pearson's correlation score, indicated a strong alignment between these two. Models that demonstrated a deeper understanding of software principles consistently outperformed others on real-world tasks.

Selection bias in MCQs format We experimented evaluating with multiple answer order permutations (follow Zheng et al. (2024a)), the result displayed significant inconsistent behavior exhibited by

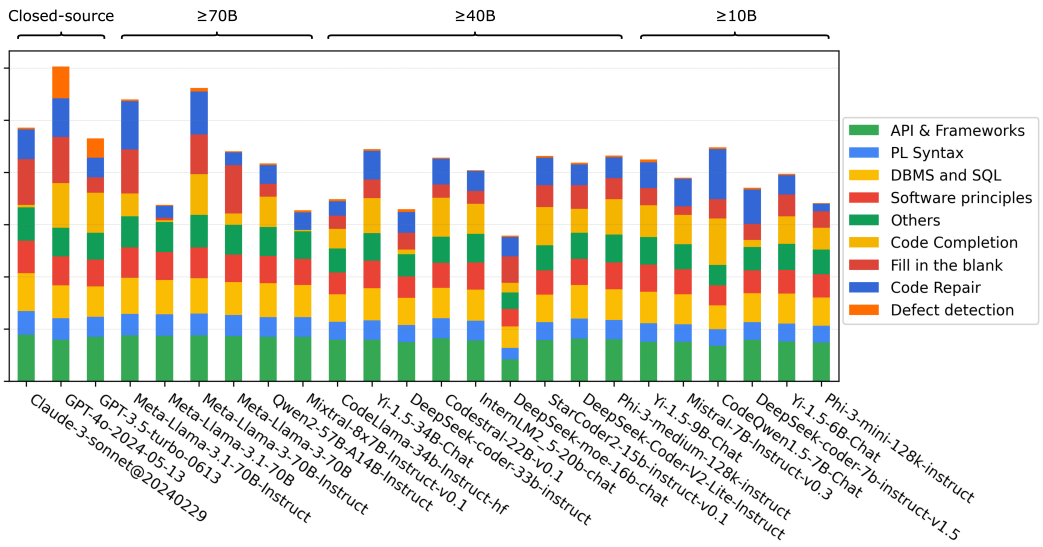

Figure 4: **CodeMMLU accuracy by task on LLMs.** While knowledge tasks are following the scaling law, real-world tasks offer more challenges to LLMs which indicate the performance of instruction tuning and data quality when evaluating on CodeMMLU.

LLMs when swapping golden answer positions. As presented in Tab. 7, the model's performance changes dramatically in each answer order configuration, which is based on the correct answer's position. The LLMs accuracy fluctuates between different permutations (i.e. DeepSeek-Coder-34B $\Delta\sigma = 36.66$), demonstrating how sensitive it can be to the structure and ordering of answers (Fig. 6).

Figure 5: **Performance Comparison between HumnaEval and MCQ Code Completion Tasks.** The performance fluctuation highlights the selection biases observed when the correct (golden) answer is moved to positions A, B, C, or D.

Models	HumanEval	Code Completion MCQ A	B	C	D
CodeLlama-7B-Python	40.48	0.00 (-40.48)	90.24 (+49.76)	14.02 (-26.46)	0.61 (-39.87)
CodeLlama-7B-Instruct	45.65	3.66 (-41.99)	1.22 (-44.43)	93.90 (+48.25)	15.85 (-29.80)
CodeLlama-13B-Python	42.89	0.61 (-42.28)	54.88 (+11.99)	70.12 (+27.23)	12.20 (-30.69)
CodeLlama-13B-Instruct	50.6	2.44 (-48.16)	68.29 (+17.69)	72.56 (+21.96)	29.88 (-20.72)
CodeLlama-34B-Python	45.11	0.61 (-44.50)	77.44 (+32.33)	70.73 (+25.62)	49.39 4.28
CodeLlama-34B-Instruct	50.79	9.15 (-41.64)	84.76 (+33.97)	65.24 (+14.45)	46.34 (-4.45)
Deepseek-Coder-7B-base-v1.5	43.2	40.85 (-2.35)	74.39 (+31.19)	64.02 (+20.82)	39.02 (-4.18)
DeepSeek-Coder-33B-base	56.1	1.22 (-54.88)	82.32 (+26.22)	75.00 (+18.90)	56.10 (0.00)
Phind-CodeLLama-34B-v2	71.95	6.10 (-65.85)	90.85 (+18.90)	75.00 (+3.05)	46.34 (-25.61)
Mixtral-8x7B-Instruct-v0.1	40.2	22.56 (-17.64)	74.39 (+34.19)	71.95 (+31.75)	63.41 (+23.21)

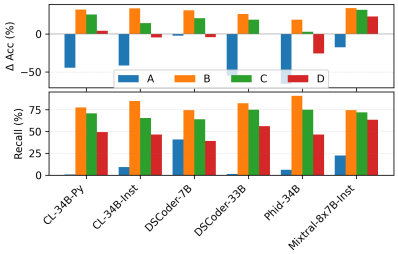

Figure 6: **Task-Specific Accuracy and Performance Fluctuations Across Answer Options** Models exhibit marked fluctuations in accuracy depending on the position of the correct answer in Code Completion in CodeMMLU. Revealing the bias and inconsistencies in related coding multiple-choice question (MCQ) task and how sensitive LLMs are to answer ordering.

Disagreement between Traditional Benchmarks and CodeMMLU A notable finding from our experiments is the discrepancy between performance on traditional benchmarks when comparing side-by-side the question HumanEval, and our code completion test set, which is HumanEval MCQ-based format. Many LLMs that perform well on HumanEval do not consistently translate their success to CodeMMLU. For instance, when comparing sample questions from HumanEval with their MCQ equivalents, the number of cases where models answered both correctly or both incorrectly was surprisingly low (Fig. 7).

This lack of alignment suggests that traditional benchmarks might overestimate a model's understanding by focusing too narrowly on code generation. The MCQ format in CodeMMLU forces models to engage with more complex reasoning and contextual understanding, exposing weaknesses that remain hidden in generative tasks.

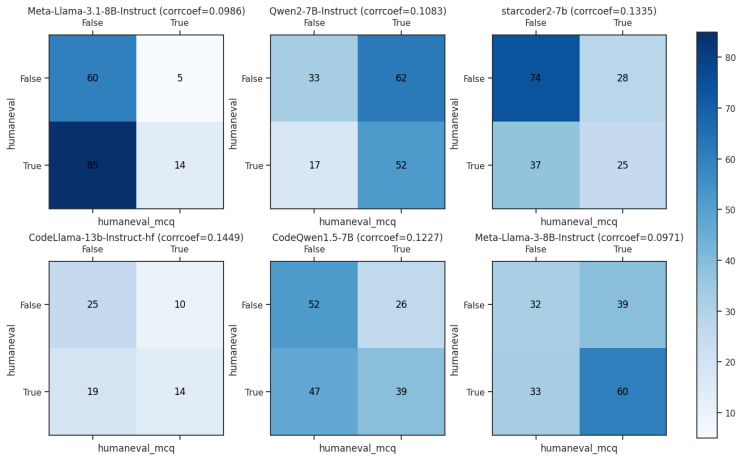

Figure 7: **CodeMMLU Task accuracy**.

5 Conclusions, Limitations, and Future Work

In this work, we introduced CodeMMLU, a comprehensive and scalable benchmark designed to evaluate large language models' (LLMs) capabilities across a wide range of software knowledge and real-world programming tasks. Our experiments highlighted the benchmark's key advantages, including its cost-effectiveness, scalability, and extensive task coverage. The insights gained revealed a strong correlation between software knowledge and real-world task performance, demonstrating that models with deeper comprehension outperform those relying purely on probabilistic generation.

Additionally, CodeMMLU provides more accurate and detailed rankings of LLMs, particularly in open-source models, where significant reordering of performance was observed. The benchmark also revealed inconsistencies in model comprehension when compared to traditional evaluations like HumanEval, emphasizing the need for more robust benchmarks that go beyond simple code generation.

Limitations. While CodeMMLU offers a broad and diverse evaluation, there are some limitations. First, the MCQ format, though effective at testing comprehension, might not fully capture creative aspects of code generation or models' ability to optimize code. Second, the current scope of languages and tasks could be expanded to include more specialized domains or additional programming languages to better assess models' versatility.

Future Work. Looking forward, we plan to release CodeMMLU as an open-source benchmark for the research community. This release will include the full dataset, along with tools for automated evaluation, allowing for widespread adoption and further improvements. Future updates will focus on adding more complex tasks, refining the balance between real-world scenarios and theoretical knowledge, and incorporating user feedback to make the benchmark even more robust for next-generation LLMs.

REFERENCES

The claude 3 model family: Opus, sonnet, haiku. URL https://api.semanticscholar.org/CorpusID:268232499.

Qwen2 technical report. 2024.

01. AI, :, Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, Kaidong Yu, Peng Liu, Qiang Liu, Shawn Yue, Senbin Yang, Shiming Yang, Tao Yu, Wen Xie, Wenhao Huang, Xiaohui Hu, Xiaoyi Ren, Xinyao Niu, Pengcheng Nie, Yuchi Xu, Yudong Liu, Yue Wang, Yuxuan Cai, Zhenyu Gu, Zhiyuan Liu, and Zonghong Dai. Yi: Open foundation models by 01.ai, 2024. URL https://arxiv.org/abs/2403.04652.

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, et al. Santacoder: don't reach for the stars! *arXiv preprint arXiv:2301.03988*, 2023.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Nghi DQ Bui, Hung Le, Yue Wang, Junnan Li, Akhilesh Deepak Gotmare, and Steven CH Hoi. Codetf: One-stop transformer library for state-of-the-art code llm. *arXiv preprint arXiv:2306.00029*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024. URL https://arxiv.org/abs/2403.04132.

Everton da Silva Maldonado, Emad Shihab, and Nikolaos Tsantalis. Using natural language processing to automatically detect self-admitted technical debt. *IEEE Transactions on Software Engineering*, 43(11):1044–1062, 2017.

Yangruibo Ding, Zijian Wang, Wasi Uddin Ahmad, Hantian Ding, Ming Tan, Nihal Jain, Murali Krishna Ramanathan, Ramesh Nallapati, Parminder Bhatia, Dan Roth, and Bing Xiang. Crosscodeeval: A diverse and multilingual benchmark for cross-file code completion. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton

Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi,

Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

Lingyue Fu, Huacan Chai, Shuang Luo, Kounianhua Du, Weiming Zhang, Longteng Fan, Jiayi Lei, Renting Rui, Jianghao Lin, Yuchen Fang, Yifan Liu, Jingkuan Wang, Siyuan Qi, Kangning Zhang, Weinan Zhang, and Yong Yu. Codeapex: A bilingual programming evaluation benchmark for large language models. *ArXiv*, abs/2309.01940, 2023. URL https://api.semanticscholar.org/CorpusID:261530384.

GeeksforGeeks. GeeksforGeeks | Quiz Hub: Test Your Knowledge, 2024. URL https://www.geeksforgeeks.org/quizzes/.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024a.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I. Wang. Cruxeval: A benchmark for code reasoning, understanding and execution, 2024b. URL https://arxiv.org/abs/2401.03065.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with apps. *NeurIPS*, 2021.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Mapping language to code in programmatic context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1643–1652, 2018.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Mohammad Abdullah Matin Khan, M Saiful Bari, Xuan Long Do, Weishi Wang, Md Rizwan Parvez, and Shafiq Joty. xcodeeval: A large scale multilingual multitask benchmark for code understanding, generation, translation and retrieval. *arXiv preprint arXiv:2303.03004*, 2023.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pp. 18319–18345. PMLR, 2023.

Linyi Li, Shijie Geng, Zhenwen Li, Yibo He, Hao Yu, Ziyue Hua, Guanghan Ning, Siwei Wang, Tao Xie, and Hongxia Yang. Inficoder-eval: Systematically evaluating the question-answering capabilities of code large language models. *arXiv preprint arXiv:2404.07940*, 2024.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Derrick Lin, James Koppel, Angela Chen, and Armando Solar-Lezama. Quixbugs: a multi-lingual program repair benchmark set based on the quixey challenge. *Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, 2017. URL https://api.semanticscholar.org/CorpusID:7158771.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods, 2022. URL https://arxiv.org/abs/2109.07958.

Chenxiao Liu and Xiaojun Wan. Codeqa: A question answering dataset for source code comprehension. *arXiv preprint arXiv:2109.08365*, 2021.

Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, and Li Zhang. Exploring and evaluating hallucinations in llm-powered code generation. *arXiv preprint arXiv:2404.00971*, 2024a.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36, 2024b.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024a. URL https://arxiv.org/abs/2402.19173.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024b.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. *arXiv preprint arXiv:2102.04664*, 2021.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023.

Alexandre Matton, Tom Sherborne, Dennis Aumiller, Elena Tommasone, Milad Alizadeh, Jingyi He, Raymond Ma, Maxime Voisin, Ellen Gilsenan-McMahon, and Matthias Gallé. On leakage of code generation evaluation datasets. *arXiv preprint arXiv:2407.07565*, 2024.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo

Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Fabio Palomba, Dario Di Nucci, Michele Tufano, Gabriele Bavota, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. Landfill: An open dataset of code smells with public evaluation. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pp. 482–485. IEEE, 2015.

Pouya Pezeshkpour and Estevam Hruschka. Large language models sensitivity to the order of options in multiple-choice questions. pp. 2006–2017, 01 2024. doi: 10.18653/v1/2024.findings-naacl.130.

Nikhil Pinnaparaju, Reshinth Adithyan, Duy Phung, Jonathan Tow, James Baicoianu, Ashish Datta, Maksym Zhuravinskyi, Dakota Mahan, Marco Bellagente, Carlos Riquelme, et al. Stable code technical report. *arXiv preprint arXiv:2404.01226*, 2024.

Ruchir Puri, David Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian T Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/a5bfc9e07964f8dddeb95fc584cd965d-Paper-round2.pdf.

Mirza Masfiqur Rahman and Ashish Kundu. Code hallucination. *arXiv preprint arXiv:2407.04831*, 2024.

Joshua Robinson, Christopher Michael Rytting, and David Wingate. Leveraging large language models for multiple choice question answering, 2023. URL https://arxiv.org/abs/2210.12353.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

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, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. URL https://arxiv.org/abs/2308.12950.

SanFoundry. SanFoundry, 2024. URL https://www.sanfoundry.com/.

Ben Shneiderman and Richard Mayer. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Parallel Programming*, 8:219–238, 06 1979. doi: 10.1007/BF00977789.

Zayne Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning, 2024. URL https://arxiv.org/abs/2409.12183.

M.-A. Storey. Theories, methods and tools in program comprehension: past, present and future. In *13th International Workshop on Program Comprehension (IWPC'05)*, pp. 181–191, 2005. doi: 10.1109/WPC.2005.38.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge, 2019. URL https://arxiv.org/abs/1811.00937.

W3Schools. W3Schools.com, 2024. URL https://www.w3schools.com/quiztest/.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators, 2023a. URL https://arxiv.org/abs/2305.17926.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*, 2021.

Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*, 2023b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. Measuring program comprehension: A large-scale field study with professionals. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 584–584, 2018. doi: 10.1145/3180155.3182538.

Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pp. 1–10, 2022.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. Repocoder: Repository-level code completion through iterative retrieval and generation. *arXiv preprint arXiv:2303.12570*, 2023.

Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. On large language models' selection bias in multi-choice questions. *arXiv preprint arXiv:2309.03882*, 2023.

Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. Large language models are not robust multiple choice selectors. In *International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=shr9PXz7T0.

Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. Large language models are not robust multiple choice selectors, 2024b. URL https://arxiv.org/abs/2309.03882.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*, 2024c.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877*, 2024.

A APPENDIX

A.1 PROMPT COMPARISON

We experimented on 48 LLM models from over 12 family with 3 different settings, namely: Zero-shot, Few-shot and Chain-of-Thought (CoT). The overall trend can be demonstrated through GPT-4o and MetaLlama 3 70B (Fig. 8), the SOTA models in closed-source and open-source families. The LLMs show a significant decrease in performance in the Chain-of-Thought prompt setting compare to few-shot and zero-shot settings. As concluded in Sprague et al. (2024) final result, CoT bring extra "thinking" step into the original task which only effective on task involved math or logic while the result on benchmark like MMLU are identical with and without CoT prompting. The decreasing that happened mostly in CodeLLM knowledge test set align with Sprague et al. (2024) conclusion of the inefficient of CoT for non-reasoning task.

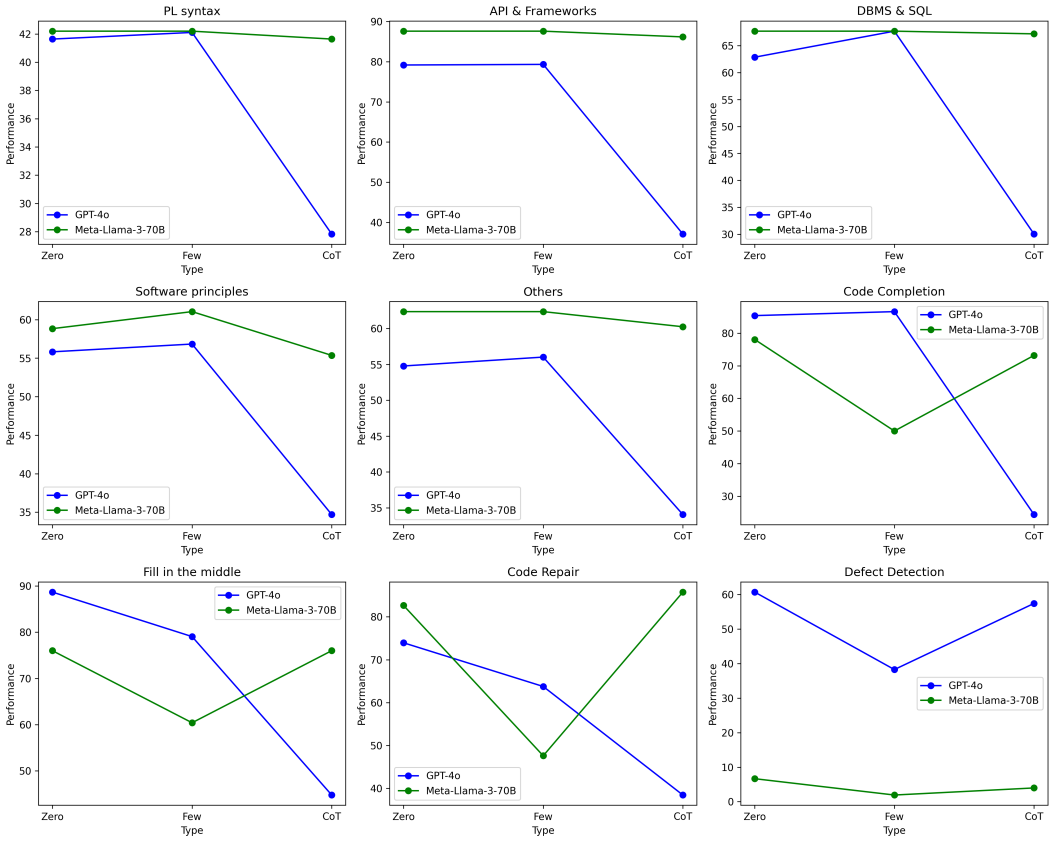

Figure 8: **Comparison between GPT4o and Meta Llama-3 70B on various prompt settings**. We experiment with zero-shot, 1-shot, and CoT prompt configuration, where the result indicates the ineffectiveness of CoT in boosting the models' performance. Comparing to zeroshot config, 1-shot prompt slightly increase the performance in knowledge tasks but falls shorter in real tasks.

A.2 MULTIPLE-CHOICE QUESTION ANSWERING FILTERING

A.2.1 KNOWLEDGE SET: RAW DATA FILTER BASED LLM

TODO: This section describe how knowledge subset is filtering out

This includes:

- **Incomplete Data:** Instances that lack a complete problem statement, code snippet, or solution are discarded.

- **Media Content:** Data containing images, videos, or other non-textual elements are excluded to maintain consistency and ensure that the benchmark focuses solely on text-based coding tasks.
- **Irrelevant or Redundant Content:** Any data that does not contribute to the assessment of programming knowledge or coding ability is filtered out.

A.2.2 KNOWLEDGE SET: CONTROLLING DATA CHALLENGING

To ensuring the knowledge set meets the quality and challenging criteria, in order to provide a reliable measure of LLM performance in the coding and programming domain. we pass the filtered subset through a meticulous process to enhance the robust and comprehensive.

- **Common Errors Identification:** We scan the dataset for common errors or issues that could lead to misleading assessments, such as ambiguous problem statements or incorrect solutions.
- **Difficulty Level Assessment:** We evaluate the difficulty level of each instance, ensuring that the tasks are neither too trivial nor exceedingly difficult. Instances deemed too easy for LLMs are removed, as they do not contribute significantly to evaluating the model's advanced problem-solving abilities.

  This process is done after we benchmark through variety of LLMs to cherry pick most common correct instances, and then spend manual effort to validate them.
- **Final Filtering:** Any data that fails to meet our stringent quality criteria is filtered out, resulting in a final dataset that is both high-quality and well-suited for benchmarking.

By focusing on a wide range of software design principles subject, we target the benchmark to set a new standard for evaluating LLMs in this critical area of AI development.

A.3 CODEMMLU ANALYSIS

TODO: - Visualize CodeMMLU - Analysis CodeMMLU properties:

- Question Length
- Difficulty
- Filter/Execution statistic