OIBench: Benchmarking Strong Reasoning Models with Olympiad in Informatics

Yaoming Zhu[1], **Junxin Wang**[2], **Yiyang Li**[3], **Xin Ding**[4], **Lin Qiu**[3], **ZongYu Wang**[3], **Jun Xu**[3], **Xuezhi Cao**[3], **Yuhuai Wei**[3], **Mingshi Wang**[3], **Xunliang Cai**[3], **Rong Ma**[4]

[1]AGI-Eval, [2]Beijing Normal University, [3]Meituan, [4]Shanghai Jiao Tong Univerisity

Abstract

As models become increasingly sophisticated, conventional algorithm benchmarks are increasingly saturated, underscoring the need for more challenging benchmarks to guide future improvements in algorithmic reasoning. This paper introduces OIBench, a high-quality, private, and challenging olympiad-level informatics dataset comprising 250 carefully curated original problems. We detail the construction methodology of the benchmark, ensuring a comprehensive assessment across various programming paradigms and complexities, and we demonstrate its contamination-resistant properties via experiments. We propose Time/Space Completion Curves for finer-grained efficiency analysis and enable direct human-model comparisons through high-level participant evaluations. Our experiments reveal that while open-source models lag behind closed-source counterparts, current SOTA models already outperform most human participants in both correctness and efficiency, while still being suboptimal compared to the canonical solutions. By releasing OIBench as a fully open-source resource [1] , we hope this benchmark will contribute to advancing code reasoning capabilities for future LLMs.

1 Introduction

Large language models have demonstrated remarkable capabilities in algorithmic reasoning and complex problem-solving, and significantly empowered human productivity in these tasks, assisting programmers in designing algorithms, optimizing computational efficiency, and solving high-level programming challenges. As models become increasingly sophisticated, conventional algorithm benchmarks like HumanEval [1] and MBPP [2] are saturated, with most state-of-the-art LLMs achieving solve rates exceeding 90%. The saturation underscores the need for more challenging benchmarks to guide future improvements in algorithmic reasoning [3, 4].

Recent benchmarks proposed more difficult coding datasets by collecting data from open-source platforms, such as Codeforces [5, 6], USACO [7], and LeetCode [8, 9]. However, their test data faced significant exposure risks given that modern LLMs utilize large-scale Internet data as their pre-training corpus. In particular, many large models will extensively crawl data from code competitions to improve performance [10].

The emergence of recent advanced test-time scaling models, such as DeepSeek-R1 [11] and OpenAI-O1 [12], has further elevated the reasoning capabilities of LLMs, especially on complex problems. Their technical reports often compare the performances with human, as CodeElo's [6] online rating strategy. However, these evaluation schemes require models to participate in online competitions over months. Moreover, human participants' varying numbers and skill levels in different sessions

[1] `https://huggingface.co/datasets/AGI-Eval/OIBench`

Preprint.

introduce data noise and reproducibility challenges. Although EffiBench introduces execution time and memory usage metrics, these remain coarse-grained average scores that obscure critical performance variations across problem types.

To address the limitations, we present OIBench, a high-quality, private, and challenging informatics olympiad-level dataset comprising 250 carefully curated original problems. The OIBench is bilingual; each problem is presented in Chinese and English with rigorous verification confirming their absence from public repositories before release. For each problem, we provide multiple forms of evaluation, including assessments in different programming languages, evaluations assisted by pseudocode, and code comprehension tasks, thereby enabling a comprehensive evaluation of algorithmic reasoning abilities. We also conduct experiments to show that supervised training cannot solve such in-distribution challenges, preventing leakage risks from the dataset open-source to future dataset updates. To enable finer-grained efficiency analysis, we propose Time/Space Completion Curves, which visualize algorithmic optimality across varying time/space limits, surpassing the granularity of prior normalized averages. In addition, we invite participants of high-level programming competitions to solve a subset of OIBench, enabling direct comparisons between LLMs and human performance. To foster reproducibility, OIBench is fully open-source, releasing not only standard components (problems, test cases, difficulty tags) but also canonical solutions, baseline model responses, and detailed reproduction costs.

We list the OIBench's contributions as follows:

1. OIBench features highly private and challenging data, collected exclusively from Olympiad in Informatics competition coaches. We verify through search engine queries that none of these problems are publicly accessible before release. Furthermore, our experiments demonstrate OIBench's resilience against data leakage: conventional models fail to acquire substantial problem-solving capabilities on our benchmark even when fine-tuned with in-distribution data.
2. We measure LLM's ability to solve Olympiad-level coding problems via OIBench, where strong reasoning models significantly outperform conventional LLMs. We develop additional pseudo-code evaluation based on the original problems to systematically assess different models' coding capabilities. Our paper presents an in-depth analysis of these evaluations.
3. We also compare the SOTA models' performance with International Collegiate Programming Contest (ICPC)-level human participants on a subset of OIBench, and conduct a comprehensive analysis. We find that current strong reasoning models outperform average ICPC-level contestants in both time and space complexity when solving algorithmic problems, even reaching optimal levels.
4. OIBench is fully open-source, releasing not only standard components (problems, labels, difficulty levels, test cases) but also canonical solutions. Considering the accessibility and reproducibility challenges of closed-source models, we further provide all baseline model responses and detailed reproduction costs.

2 Related Work

Reasoning Benchmarks. Since recent studies [13] introduced the concept of Chain-of-Thought (CoT), the research community has regarded it as a plausible way towards artificial general intelligence [14, 15]. Based on CoT, researchers developed various benchmarks on measuring the LLM's reasoning abilities, including mathematics (GSM8K [16], MATH [17]), commonsense (HotpotQA [18]) , logics [19] , coding and other topics (CriticBench [20]).

Recent studies debated whether existing benchmarks properly assessed reasoning capabilities. While some attributed improvements to memorization [3], others attributed insufficient challenging problems in benchmarks [4]. The community developed new benchmarks to address these gaps: Sys2Bench (multi-task reasoning) [21], CriticBench (critique/correction) [20], LogicVista (visual logic) [22], LiveBench (contamination-free evaluation) [23], LogicGame (rule-based planning) [24], and QRData (statistical reasoning) [25]. These collectively provided systematic evaluation frameworks while revealing LLMs' reasoning strengths and weaknesses.

Coding Benchmarks, such as HumanEval [1] and CoderEval [26], offered the standard for evaluating the code capabilities of models via programming problems. More complex benchmarks, such as CodeElo [6] and USACO [7], introduced more challenging, competition-level problems.

Benchmark	Difficulty	Avg. Words per Prob.	Avg.Size of Test per Prob.	Avg. Lines of Canonical Solution	Non-symbolic	Time/Space Metrics	Language	Source
HumanEval[1]	⋆	67.7	65.4k	8.7	✗	✗	en	Private
CodeContests[5]	⋆⋆	344.5	91.7k	52.5	✓	✗	en	Codeforces
USACO[7]	⋆⋆⋆	452.9	59.8k	28.7	✓	✗	en	USACO
CodeElo[6]	⋆⋆	147.9	-	-	✓	✗	en	Codeforces
LiveCodeBench[8]	⋆⋆	271.5	3.85M	-	✗	✗	en	LeetCode, *etc.*
EffiBench[9]	⋆	212	82.5k	16	✗	✓	en	LeetCode
OIBench	⋆⋆⋆⋆	225.3	37.5M	75.2	✓	✓	zh/en	Private

Table 1: Basic statistics, information between ours OIBench and other code benchmarks.

EffiBench [9] focused on evaluating the efficiency of the generated code, while the RACE benchmark further extended the evaluation dimensions by considering four key aspects[27, 28]. CRUXEval [29] and EquiBench [30] assess LLM performance in code understanding tasks. PseudoEval [31] evaluates models' coding ability via pseudocode. Existing static benchmarks were susceptible to data contamination, compromising the reliability and generalizability of evaluation results. DynaCode [32] effectively mitigated data contamination by generating new code samples [33]. Additionally, LiveCodeBench [8] provided a comprehensive, low-contamination code evaluation platform, enhancing the timeliness of benchmarks.

3 OIBench Construction

The olympiad-level informatics problems are originally collected from ICPC competition team coaches. These coaches, on average, have 20 years of experience in coaching university ICPC teams and high school Olympiad Informatics (OI) teams in China. Their expertise in creating high-difficulty algorithmic problems is instrumental in our data collection process.

To ensure the quality of these challenging algorithmic problems in our study, we ask the coaches either to select problems from their private repositories or to compose new problems that meet our stringent criteria as follows:

1. **Originality and Confidentiality**: To prevent data leakage, each problem must be original and unpublished, whether online or in print-media. Importantly, minor adaptations of existing problems, such as changing numerical values or rephrasing the problem statement without altering the underlying logic, are not considered original.
2. **Difficulty**: To ensure that the problems provided by the coaches are difficult enough to effectively assess frontier models' reasoning capabilities, We require coaches to assign difficulty labels to problems based on the Codeforces difficulty rating. In addition, each problem must be labeled with its corresponding competition level. Recognizing the subjective nature of human-based difficulty classification, we utilize LLMs in our acceptance process to verify the problem's difficulty. Specifically, we refer to the difficulty control method proposed by [34], where we employ four models to solve each problem, and we accept a problem only if no more than one of these models solve it correctly.
3. **Robust Test Cases and Verified Canonical Solutions**: Effective test cases are essential for accurately evaluating the participants' solutions. Usually, by including test scenarios against large input sizes and resource-intensive operations, test cases can reveal efficiencies and potential bottlenecks in time and space utilization for a solution program. Besides, high-quality test cases shall also examine the program's resilience against corner cases. Hence, we require the coaches to craft test cases that meet our standard listed in the Appendix. Additionally, coaches are required to provide canonical solutions in **C++** to confirm that all problems are solvable and test cases are correct. We also verify the correctness of the canonical programs by executing all test cases.

We employ six ICPC student participants to review the quality of the questions after the collection process, ensuring that each question is *solvable*, *clearly described*, includes *appropriate sample input/output*, and that the formulas/tables in the problem statement adhere to TEX/Markdown syntax.

After verifying the quality of the Chinese problem statements, we recruit professional translators with related work experience to translate the problems into English, where both native English and Chinese translators are involved in the translation process for each problem to ensure translation

quality. We include background profiles, work duration, and wage information from all dataset annotation employees in the appendix A.10. We give an example problem in Appendix Fig. 4.

We present a comparative analysis between our benchmark and competing benchmarks in Table 1. The difficulty levels are calibrated based on the accuracy of GPT-4o, shown in Appendix Table 4. OIBench significantly exceeds competing benchmarks in terms of problem difficulty, and test case size. Furthermore, we ensure that at the time of dataset release (May 2025), none of the problems can be directly retrieved from the Internet.

3.1 Anti-Contamination

A challenge for benchmark design is the in-distribution contamination problem [35], as modern LLMs typically scrape Internet data for pre-training and fine-tuning, potentially including contents with the same domain of the benchmark (e.g. the training set of GSM8k). The contamination not only renders leaked problems ineffective as test cases, but also artificially inflates performance scores on benchmarks of the same domain without actual improvements on the corresponding abilities[36–39].

To assess OIBench's robustness against the contamination, we conducted experiments using 100 held-out C++ problems with official solutions as training targets. We mix these problems with 10,000 standard supervised fine-tuning samples to simulate in-distribution contamination scenarios. We train two variants: baseline models using only standard SFT data, and leaked models additionally trained on the 100 held-out problems.

We propose a Risk-Score metric to quantify how in-domain data might lead to contamination. The metric balances absolute and relative improvements by measuring progress toward perfect accuracy: Risk Score $= \frac{S_{\text{leaked}} - S_{\text{baseline}}}{1 - S_{\text{baseline}}}$, where S_{leaked} and S_{baseline} represent scores for the respective models.

We conduct the similar contamination experiments on HumanEval and LiveCodeBench to further comparison. Specifically, for HumanEval, we randomly sample 100 problems as leaked data and use the remaining problems as the test set. For LiveCodeBench, we sample 100 problems dated before July 2024 as leaked data, and evaluate on problems dated after August 2024.

Table 2 shows the Risk-Score on different size models. We see that the Risk-Score is extremely low($\leq 1.03\%$) for various models on OIBench, while remains high for other benchmarks. Since LiveCodeBench was collected online, models trained on LeetCode website data may exhibit certain score distortions on the benchmark. In comparison, models cannot obtain improvements on OIBench unless they obtain comprehensive improvements on all abilities that are related to coding, for example, from Llama3-8B to Deepseek-V3. We owe this robustness to the difficulty of OIBench, we believe that the presence of potential in-distribution data will not adversely affect OIBench's evaluation of the actual code capabilities of LLMs.

Risk Score(%)	Llama3-8B	Qwen2.5-7B	Qwen2.5-14B	Qwen2.5-72B	Qwen2.5-Coder-32B	Deepseek-V3
OIBench	0.00	0.00	0.00	1.00	1.00	1.03
HumanEval	12.50	5.56	17.65	7.14	23.08	–
LiveCodeBench	15.39	14.49	13.32	10.20	6.01	–

Table 2: Risk-Score of benchmarks on different models.

4 OIBench Learderboards

In this section, we report the evaluation results on state-of-the-art LLMs.

Evaluation Environment. We evaluate all implementations on a computing server running CentOS Linux 7.6.1810 (Core) with Linux kernel x86-64. For each language: For C++ implementations, we compile the code using g++ 11.3.0 with C++17 standard compliance and -O2 optimization flags. For JavaScript implementations, we execute the code using Node.js v16.18.1 with npm v8.19.2 for dependency management. For Java implementations, we run the programs using Java 1.8.0_45 with the Java HotSpot 64-Bit Server VM. For Python components, we employ Python 3.9 for orchestration. The environment operates under Linux kernel 5.4.0 with glibc 2.31, using Docker [40] containers with privileged hardware access. We monitor resources through system utilities (time command and /proc filesystem analysis) and configure environment variables for optimal runtime performance.

Baseline Models. We mainly report code performance of 19 state-of-the-art LLMs, including 15 instruct models and 4 base models, the details for each model is elaborted in Table in Appendix. We list the prompts for instruct or base models respectively in Appendix. All models are evaluated in zero-shot. For reasoning models, we set the maximum inference tokens to be 32,768. For each model, we evaluate its coding ability on 4 languages, namely C++/Python/Java/JavaScript.(We include the rationale for evaluating four programming languages in the Appendix.)

By default, we use greedy search sampling with temperature as 0 for all tested models for output stability, except for the DeepSeek-R1-0528 and Qwen3-32B model, where we find that its reasoning process tends to collapse in recursive patterns under greedy search. We set its temperature at 0.6 as its paper suggests [41].

4.1 OIBench Leaderboard

We list the evaluation results in Table 3, and report the confidence interval [42] in appendix A.6.

We find that reasoning models achieve an average Overall score of 24.3% on OIBench, significantly outperforming instruction-tuned models, which score around 5.4%. Among all models, Deepseek-R1-0528 and o4-mini-high rank highest, consistently outperforming across all languages and tasks. This demonstrates the reasoning models' advantage in solving complex problems and highlights OIBench's ability to evaluate reasoning and chain-of-thought capabilities, distinguishing models in terms of reasoning ability.

Closed-source models score 15.9% on average, while open-source models score 8.4%. This performance gap is consistent across both categories, as closed-source models generally have more compute resources and higher-quality training data [43].

For instruction-tuned models, coding performance on OIBench strongly correlates with base model capabilities. Given the zero-shot evaluation, this suggests that base models already possess substantial coding potential, with instruction tuning providing marginal improvements on coding. We encourage focusing on base model capabilities rather than fine-tuned variants. The exception is DeepSeek-V3-0324, which surpass all other non-reasoning models. As DeepSeek-V3-0324 employs DeepSeek-R1' s CoT for distillation, which enhances the reasoning capabilities. We provide a detailed analysis of this improvement in later sections.

Regarding programming languages, models perform over $5 - 10\%$ worse on JavaScript and Python compared to C++ and Java on average. This discrepancy likely stems from factors such as skewed data distributions and training biases, which requires further investigation. Regarding natural languages, models show minimal performance differences between Chinese and English, with Chinese performing slightly better, likely due to a latent source language preference in prompt translation.

4.2 OIBench with Pseudocode

As the previous section reports, our proposed OIBench is highly complex for non-reasoning models and base models to obtain correct answers without reasoning process. In real-world scenarios, a finer-grained analysis of LLM performance on code; simultaneously, we want to evaluate models' understanding of the reasoning process. Therefore, we employ DeepSeek-R1 to convert canonical solutions into pseudocode and use it as prompts. We ensure that the pseudocode remains language-agnostic. When evaluating problems, we provide both the problem statement and the pseudocode as prompts to test the models' problem-solving capabilities, we provide results in Table 3.

With the help of pseudocode solutions, all models show a significant performance improvement when provided with solution hints. Even for the strongest models, such as Deepseek-R1-0528 and o4-mini-high, OIBench Pseudo results in a noticeable boost in performance. This indicates that the solution hints greatly reduce the difficulty that originally relied on complex reasoning, narrowing the gap between reasoning and non-reasoning ones. This suggests that this evaluation method is more suitable for assessing models' ability to understand solutions and their code generation capabilities. At the same time, long CoT reasoning models still maintain an advantage in solution comprehension.

Furthermore, with help of pseudocode, we observe that the performance of post-trained models remains highly correlated with that of their corresponding Base models, further indicating that the code generation potential of a model is closely tied to the level of its pretraining.

Model \ Subset	OIBench					OIBench by language		OIBench Pseudo					Pseudo by language	
	Overall	C++	Java	Python	JS	En	Zh	Overall	C++	Java	Python	JS	En	Zh
Base Models														
DeepSeek-V3-base [44]	2.85	5.20	4.20	2.00	0.00	2.30	3.40	27.45	41.60	37.00	29.40	1.80	28.20	26.70
Qwen2.5-Coder-32B-base [45]	0.95	0.60	2.00	1.00	0.20	0.90	1.00	17.80	21.80	22.80	24.40	2.20	17.80	17.80
Qwen2.5-72B-base [46]	0.75	1.20	1.00	0.40	0.40	0.80	0.70	18.50	21.60	25.60	26.20	0.60	18.80	18.20
Llama-3.1-405B-base [47]	0.75	1.20	1.40	0.40	0.00	0.60	0.90	18.40	19.80	25.60	28.00	0.20	18.90	17.90
Instruction Tuned Non-Reasoning Models														
GPT-4.1 [48]🔒	12.00	12.20	13.80	11.00	11.00	11.40	12.60	24.50	27.40	29.40	19.80	21.40	26.40	22.60
DeepSeek-V3-0324 [44]	11.60	13.20	11.60	13.80	7.80	11.30	11.90	32.15	35.00	36.80	29.80	27.00	32.40	31.90
Doubao-Seed-1.6-Off [49]🔒	5.95	6.60	5.60	5.40	6.20	5.80	6.10	31.15	36.40	36.00	27.60	24.60	31.10	31.20
DeepSeek-V3-1226 [44]	4.10	4.80	4.80	3.80	3.00	3.90	4.30	32.67	38.00	36.80	31.90	24.00	32.95	32.40
Claude3.5 Sonnet [50]🔒	3.45	2.80	6.20	2.20	2.60	3.60	3.30	28.90	37.00	26.40	27.00	25.20	25.80	32.00
GPT-4o [51]🔒	2.65	1.80	4.40	2.80	1.60	2.70	2.60	21.45	21.00	23.80	23.40	17.60	19.90	23.00
Qwen2.5-Coder-32B [45]	1.90	1.00	2.40	2.80	1.40	1.40	2.40	22.90	20.80	26.40	30.40	14.00	21.30	24.50
Llama-3.1-405B [47]	1.15	1.60	1.00	1.20	0.80	1.40	0.90	28.45	28.80	32.00	35.00	18.00	29.90	27.00
Reasoning Models														
Deepseek-R1-0528 [41]	37.55	42.60	38.60	34.60	34.40	37.90	37.20	49.10	53.40	51.60	49.80	41.60	49.20	49.00
o4-mini-high [52]🔒	36.35	43.40	37.20	34.60	30.20	35.60	37.10	51.70	58.80	53.40	49.80	44.80	52.80	50.60
o3-mini-high [53]🔒	26.80	31.80	26.80	28.40	20.20	27.30	26.30	44.90	54.60	44.00	42.00	39.00	45.20	44.60
Doubao-Seed-1.6-On [49]🔒	24.30	26.20	24.00	24.00	23.00	23.40	25.20	44.35	48.60	48.60	42.40	37.80	43.10	45.60
Qwen3-32B [54]	18.95	20.60	21.40	19.20	14.60	20.40	17.50	33.35	34.40	35.40	35.20	28.40	35.60	31.10
o1 [12]🔒	15.40	22.00	21.20	6.60	11.80	15.70	15.10	32.25	44.80	36.40	18.20	29.60	30.70	33.80
QwQ-32B [54]	10.65	10.60	13.60	8.40	10.00	10.30	11.00	31.70	29.60	33.00	39.40	24.80	33.40	30.00

Table 3: Main Leaderboard in accepted rate. We adopt the AC (All Correct) rate (%) in this table, where a model must pass all test cases of a problem to be considered correct. 🔒 indicates close-source models.

4.3 Code Efficiency Metrics

OI competitions' test cases are designed to be large-scale, ensuring comprehensive coverage of edge cases and providing rigorous evaluation of algorithmic time/space complexity. OIBench follows this standard with large-scaled test cases. Conventional benchmarks typically employed time/space thresholds as acceptance criteria, which suffer from two limitations: (1) they only coarsely approximate whether solutions meet specific complexity classes, (2) threshold selection can be subjective, where different threshold leads to performance fluctuation.

EffiBench [9] introduced efficiency metrics based on runtime/space ratios to canonical solutions, however, this metrics remain insufficiently intuitive: (1) it reports a single numerical value that fails to capture temporal distribution of different LLMs, (2) the metric becomes unreliable when measuring low-complexity but incorrect solutions, which can be frequent in LLMs' generation.

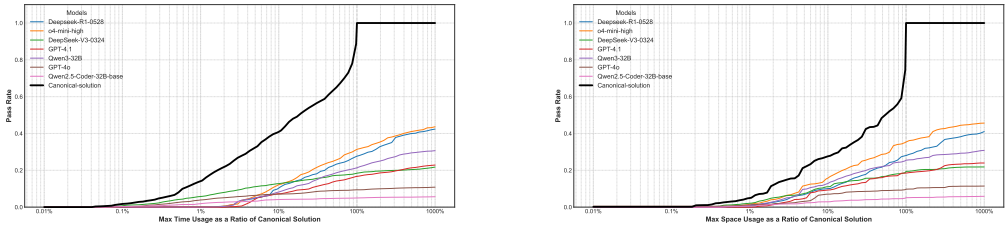
Our paper presents a fine-grand efficiency evaluation metric, namely the **Time/Space-Completion Curve**, as shown in Figure 1. This curve is essentially the cumulative distribution of solving time/space, offering significant advantages, where the X-axis represents time/space threshold (in ratio to the canonical solution) on log-scale, while the Y-axis is the fraction of solved test cases under the threshold.

Note that visualizing all models would introduce visual clutter; therefore, we primarily visualize representative models in the subsequent sections. We also report the area-under-curve in Appendix Tab 6 for a more quantitative comparison. The analysis reveals several key insights:

1. The models' algorithms underperform the canonical solution in both time and space complexity. Specifically, even after reaching 100% of canonical usage, all models still exhibit growth, suggesting potential for further inprovment in algorithmic efficiency.

2. o4-mini-high and DeepSeek-R1-0528 generally demonstrates the best time and space optimization. However, in the low time region (less than 10% of canonical usage), both models performs worse than models such as GPT-4o and DeepSeek-V3. This suggests that the algorithmic design of non-reasoning models may perform better on simpler test cases, as they output naive and inefficient algorithms.

3. Additionally, conventional LLM models are generally inefficient in their algorithmic design, while models with long CoT exhibit a significant advantage in optimizing time and space complexity. This underscores the strength of reasoning models in handling complex tasks more efficiently than instruction-tuned models.



(a) Time Completion Curve

(b) Space Completion Curve

Figure 1: Time/Space Completion Curves. Note that in the figure, we employ the test case pass rate rather than the AC rate, as the former provides more fine-grained visualization.

4.4 Test-Time Compute Comparison

Since test-time computation has become a paradigm to improve LLMs' performance on complex tasks [55], recent models utilize long Chain-of-Thought as the main test-time scaling technique. Given that similar methods incur substantial computational budget during inference, researchers typically examine the trade-off between model computational cost and performance. We report the distribution of model performance versus the number of inference tokens[2] in Figure 2. o4-mini-high demonstrates the best reasoning efficiency, solving more problems correctly with fewer tokens, while DeepSeek-R1-0528 takes more inference budgets to achieve similar pass rate. Qwen3-32B exhibits pass rates similar to GPT4.1, respectively, but require significantly more tokens for reasoning. Among non-reasoning models, DeepSeek-V3-0324 and GPT-4.1 achieves the best performance while also having the longest reasoning token count, suggesting that they have learned certain characteristics of CoT long-chain reasoning during the distillation learning of reasoning models like R1.

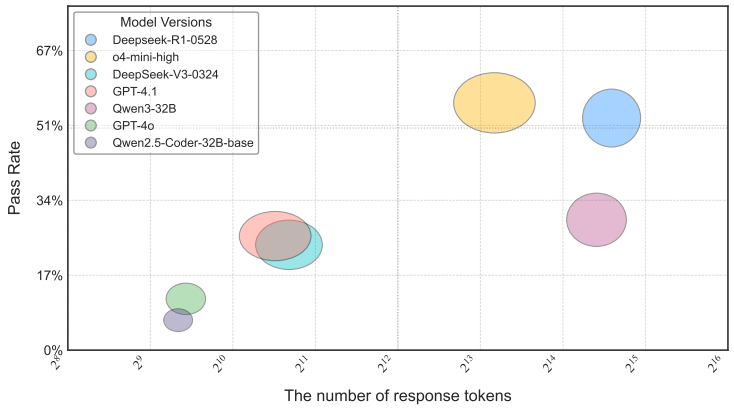


Figure 2: Inference budgets measured by tokens *vs.* pass rate. The size of the bubble is proportional to the standard deviations on the corresponding axes.

5 Comprehensive Analysis

5.1 Human-comparable Evaluation

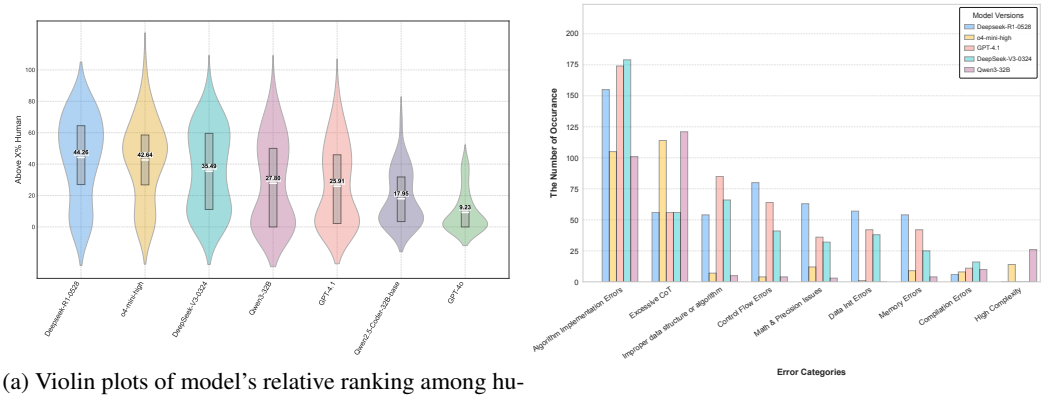
CodeElo [6] was the first study that provided an LLM-human performance comparison in code via an online code competition site CodeForces. However, online competition benchmarks exhibits

[2]For close-sourced models, we report the token usage from its API return value.

certain limitations. Their use of human Elo ratings from contests up to **six months** prior introduces a temporal misalignment, potentially leading to outdated comparisons, and difficulty in reproducibility. Additionally, their construct the Elo system via randomly sampling 250 user scores from the Codeforces website, leading to imprecision.

OIBench further conducts a systematic comparison between LLMs and human programmers via introducing static human performance metrics. Specifically, we curate a set of 44 programming problems from OIBench and recruit six ICPC-level contestants to complete them under controlled conditions, subsequently reporting aggregated results. To enhance reproducibility and facilitate deeper analysis, we release the anonymized solution records alongside the benchmark dataset, enabling the research community to rigorously examine the performance disparities between humans and large language models in code generation tasks. For each model, we report its ranking among all human participants via violin plots, as shown in Fig 3a, where the number indicates the average relative ranking for each model.

Our ranking follows the OI competition rules: Each submission is first evaluated on the basis of the number of passed test cases, with higher numbers resulting in better rankings. When submissions have the same number of test cases passed, the ranking is determined by execution time, where shorter times lead to higher rankings. For human participants, we consider their last submission as the final answer.



(a) Violin plots of model's relative ranking among human participants.

(b) The error type for each model.

Figure 3: Comparison of model ranking and error types.

We observe distinct performance patterns in the violin plots of models' relative rankings among human participants:

1. Valley-type: These models consistently rank low across most problems, typically outperforming less than 20% of human participants. All conventional non-reasoning LLMs exhibit this pattern. On high-difficulty problem sets, these models can only solve a limited number of test cases and fail to reach competitive problem-solving levels.
2. Bimodal-type: These models perform well on certain problems (e.g., surpassing around 50% of human participants) but poorly on others. Most models in this category are capable of autonomous long-CoT reasoning. The violin plots consistently show a bimodal distribution, which may reflect the fundamental characteristics of the training-inference paradigm. The presence of two peaks indicates that while those models' CoT can effectively solve certain types of problems, but they still lack more generalizable reasoning capabilities.
3. Balanced-type: This pattern features a more uniform distribution of rankings across tasks. Only advanced reasoning models (i.e., DeepSeek-R1-0528 and o4-mini-high) exhibit this pattern.

The contrast between the bimodal-type and balanced-type distributions is particularly noteworthy. The former trend highlights a potential limitation of many current long CoT training paradigms—models tend to learn strategies effective for only a subset of problem types. In contrast, the reasoning capability of o4-mini-high appears more human-like in its generality and consistency. We strongly advocate for leaderboard to include model-versus-human rating comparisons. Such visual and

statistical assessments help the research community develop a more nuanced understanding of model generalization and alignment with human capabilities.

5.2 Qualitative Error Analysis

We perform qualitative analysis of error patterns across multiple models, categorizing algorithmic erros or flaws. With assistance from DeepSeek-R1 and validation by an ICPC-level programmer, we annotate and visualize error distributions of serveral models in Figure 3. We also analyze the impact of different algorithm tags on model performance in the Appendix.

For most reasoning models, the primary cause of failure is that the Chain-of-Thought (CoT) becomes too long and cannot complete reasoning within the limited token constraint. Given our relatively large maximum reasoning length setting (32,768), we observe that these failed long reasoning chains exhibit noticeable recursive patterns. We suggest that future improvements in reasoning efficiency for CoT and mitigation of recursive patterns will be crucial for enhancing models' reasoning capabilities, as our previous analysis shows that o4-mini-high achieves the highest CoT reasoning efficiency and obtains superior results. Additionally, incorrect algorithm implementation and inefficient algorithm selection remain major obstacles affecting model performance in reasoning tasks. While issues such as program compilation errors and data processing errors occur less frequently, they still exist.

6 Conclusion

This paper introduces OIBench, a verified, private, and challenging benchmark designed to evaluate LLMs on Olympiad-level algorithmic problems. Our contributions are as follows.

First, OIBench is highly challenging even for frontier reasoning models. Existing benchmarks increasingly lose the ability to differentiate models' reasoning abilities due to saturation problems. In contrast, reasoning models substantially outperform non-reasoning models on OIBench, highlighting the benchmark's strong discriminative power.

Secondly, our Time/Space Completion Curves based on OIBench reveal considerable room for improvement in the reasoning efficiency of current models, particularly through optimizing algorithms in terms of time and space complexity.

Thirdly, human-model comparisons on OIBench show that a gap remains between LLMs and top-tier human players. Though frontier reasoning models surpass the majority of ICPC-level competitors, they still lag far behind top-tier human players. However, two SOTA models exhibits a distinct pattern compared to others. We encourage further research into human-comparable analysis to gain deeper insights.

Last but not least, for reasoning models, we suggest that optimizing the efficiency of Chain-of-Thought is a crucial direction for future advancements in reasoning capabilities. By releasing OIBench as a fully open-source resource benchmark (including problems, test cases, and canonical solutions), we aim to help advance the development of LLMs' reasoning abilities.

A Appendix

A.1 Limitations and Further Studies

Despite rigorous data cleaning procedures and verification through search engines, the dataset may still carry potential risks of internet leakage. Additionally, we cannot completely ensure the absolute originality of the coaching questions in terms of problems' paradigms, as some might be adapted from existing problem types.

The observed performance disparity between models implemented in C++/Java and those in Python/JS remains unexplained and warrants further exploration from perspective of model training.

Our anti-contamination experiments excluded reinforcement learning (RL) training components. Standard practice dictates that internet-sourced pre-training data is typically excluded from RL training phases. However, emerging pretraining methodologies like pretrain with RL [56] could potentially introduce contamination risks following OIBench's open-source release.

A.2 Example Problem

Problem Description and Details

Algorithm Tags: Monotonic Stack, Mathematics **Difficulty:** Div3 Level

Problem:

You are given a permutation of length n and an integer k. In a sequence, an element is called a key element if it is strictly greater than all the elements before it.

For all subsequences, if the number of key elements in the subsequence is exactly k, then $\texttt{ans} += (-1)^{\text{len(subsequence)}}$, where len(subsequence) represents the length of the subsequence. Initially, $\texttt{ans} = 0$.

Please compute the final value of `ans`.

Data Format & Data Range

Input Format:

The first line contains two integers n and k.

The second line contains the permutation $p_1, p_2, \ldots, p_n$.

Output Format:

Output a single integer representing $\texttt{ans} \bmod 998244353$.

Data Range:

$1 \leq n \leq 10^6$

Sample Data

Input:

```
5 2
1 4 2 5 3
```

Output:

```
3
```

Canonical Solution (Partial)

```
int ksm(int x,int y){
    int ans=1;
    while(y){
        if(y&1){ans=1ll*ans*x%mod;}
        x=1ll*x*x%mod;
        y>>=1;
    }
    return ans;
}
```

Figure 4: Example problem in OIBench.

A.3 Difficulty Comparsion

We list the competitive benchmarks' results in Table 4 with GPT-4O as the evaluated model. We report the HumanEval's result from `evalplus.github.io/leaderboard.html`. We use pass@8 results for CodeElo as their paper reported. We report the CodeContests's result from [57]. For other benchmarks, we report the AC rate (*i.e.* pass@1) results from the corresponding paper.

A.4 Training details for anti-contamination experiments

We train the models using $10,000$ supervised fine-tuning (SFT) samples, mixed with 100 samples from OIBench or other cometitive benchmarks. For all models, we adopt the Megatron-LM [58] framework for training, a learning rate ranging from 5×10^{-6} to 8×10^{-6}, the Adam optimizer, and train for 3 epochs. The choice of these hyperparameters follows common practices in the open-source community for models with similar parameter and data scales, without additional hyperparameter search or special optimization.

Due to computational resource limitations, we only investigate the performance of DeepSeek-V3 on OIBench.

A.5 Derived Benchmarks

To further evaluate LLMs' code comprehension abilities under more complex scenarios, and examine how well the model understands the complex logic of competition-level code, we introduce four derived benchmarking tasks focusing on code understanding, which consists of the following subtasks:

Bugfix(BF): Correct buggy solutions by identifying and fixing errors, similar to [59].

Complete(CP): Complete partial solutions with missing code segments, similar to [60].

Translate(TS): Translate C++ solutions to equivalent Python implementations, similar to [61].

Interpret(IT): Interpret and predict outputs based on given solutions and test inputs, similar to [29].

We list the results of the derived benchmark in Table 5. Note that base models usually fail to generate valid results for comprehension tasks, therefore we exclusively evaluate the code understanding performance of instruction-tuned models.

The derived benchmark demonstrates similar overall trends to both OIBench and OIBench Pseudo, with reasoning models consistently outperforming non-reasoning models. Notably, models exhibiting strong performance on OIBench Pseudo also tend to display superior code comprehension capabilities. A particularly distinctive case is the code interpretation task, which shows greater discriminative power than other derived benchmarks and more effectively tests models' ability to simulate code execution through reasoning.

It is worth-mentioning that Deepseek-R1-0528 performs competitively on the original reasoning tasks, a notable gap emerges between it and the O4-mini-high model on the derived benchmarks. This discrepancy suggests that, despite its strong reasoning abilities, Deepseek-R1-0528 still faces challenges in following complex instructions required by code understanding tasks such as bug fixing, completion, translation, and interpretation. These results highlight that further improvements in instruction following are necessary for models like Deepseek-R1-0528 to excel in more sophisticated code comprehension scenarios.

A.6 Statistical Significance Report

We report confidence interval in Table 7, using the statistical approach from [42]. While the confidence intervals of similar models overlap, our paper does not focus on ranking closely rival models. Instead, we highlight the significant performance gaps between different model tiers (e.g., reasoning models vs. conventional instruction-tuned models) on the Leaderboard, where the differences far exceed confidence intervals. This distinction is further supported by our extensive experiments.

A.7 Computing Resources

For anti-contamination experiments, models with fewer than 14B parameters are trained on 8 A100 GPUs (80GB each), which takes around 2 hours; models with 32B to 72B parameters are trained on

Benchmarks	HumanEval	CodeContests	USACO	CodeElo	LCB	EffiBench	OIBench
GPT-4o	87.2	34.7	8.3	16.8	41.9	50.8	2.6

Table 4: Difficulty comparison of code benchmarks via GPT-4o passing rate.

Model	OIBench Derived					OIBench Derived by language	
	Overall	BF	CP	TS	IT	En	Zh
Instruction Tuned Non-Reasoning Models							
GPT-4.1🔒	24.65	18.80	22.00	34.40	23.40	29.50	19.80
DeepSeek-V3-0324	33.25	34.60	36.20	21.60	40.60	34.80	31.70
Doubao-Seed-1.6-Off🔒	23.25	19.60	35.00	32.40	6.00	24.20	22.30
DeepSeek-V3-1226	28.20	39.60	35.80	16.40	21.00	30.30	26.10
Claude3.5 Sonnet🔒	29.55	39.20	38.80	20.40	19.80	32.80	26.30
GPT-4O🔒	27.55	39.00	30.20	15.80	25.20	29.90	25.20
Qwen2.5-Coder-32B	15.45	21.60	17.20	8.60	14.40	16.60	14.30
Llama-3.1-405B	21.00	28.80	31.60	12.40	11.20	25.00	17.00
Reasoning Models							
Deepseek-R1-0528	49.00	51.80	51.00	57.40	35.80	59.20	38.80
O4-mini-high🔒	59.75	62.60	59.40	31.40	85.60	66.80	52.70
O3-mini-high🔒	55.10	61.40	49.00	30.80	79.20	61.60	48.60
Doubao-Seed-1.6-On🔒	33.05	38.20	42.40	48.00	3.60	36.00	30.10
Qwen3-32B	28.30	28.20	21.80	20.40	42.80	33.70	22.90
O1🔒	40.70	36.60	39.00	19.80	67.40	40.00	41.40
QwQ-32B	34.35	32.60	25.60	23.60	55.60	38.60	30.10

Table 5: OIBench Derived Leaderboard in accepted rate. BF, CP, TS and IT stands for Bugfix, Code Complete, Code Translate and Code Interpret respectively.

Model	**Time**	**Space**
Canonical Solution	38.48	32.14
o4-mini-high	12.69	15.00
Deepseek-R1-0528	11.65	11.80
Qwen3-32B	9.04	10.86
GPT-4.1	6.75	7.90
DeepSeek-V3-0324	9.62	8.65
GPT-4o	5.18	4.66
Qwen2.5-Coder-32B	2.77	2.27

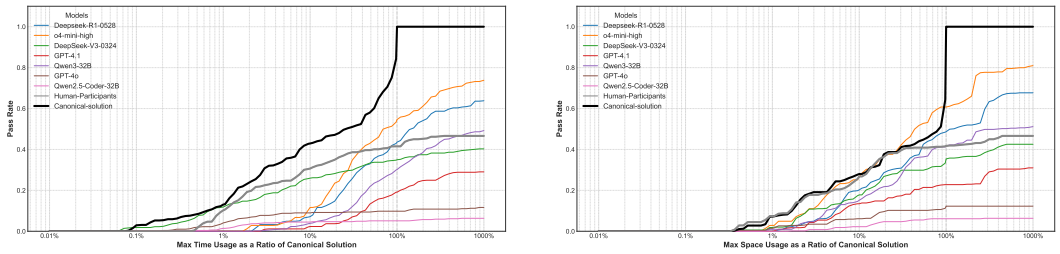
Table 6: Area-under-curve(%) of time/space completion curve

8 nodes, each equipped with 8 A100 GPUs, which takes around 5 hours. The Deepseek-V3 model is trained on 16 nodes, each equipped with 8 H20 GPUs (141GB each), which takes around 5 hours.

For model evaluation on OIBench, we list the usage in Table 7. We use vLLM [62] as the inference service. For close-sourced models, the evaluation time will fluctuate with the network and supplier services, so we **do not** provide a prediction time.

A.8 Time/Space-Completion Curve vs Human

We compare the models' performance with human contestants' in Figure 5 on 44 problems. We can see the SOTA reasoning models' solution efficiency surpass most human. The detailed analysis shown in Section 5.1. We report area-under-curve in Tab. 6.



(a) Time Completion Curve Compare with Human

(b) Space Completion Curve Compare with Human

Figure 5: Time/Space Completion Curves

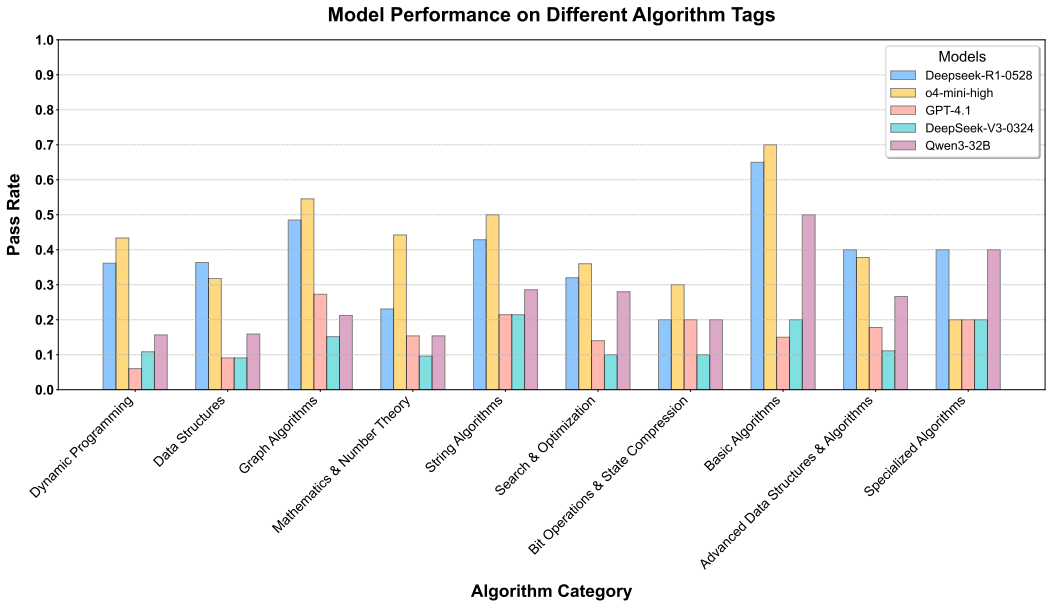


Figure 6: Models' performance on different algorithm tags

A.9 Algorithm Tags

We illustrates the performance of five models across ten distinct algorithm categories, measured by pass rates, in Fig 6. Leading models such as Deepseek-R1-0528 and o4-mini-high demonstrate balanced and robust performance across algorithmic tasks, without significant weaknesses. Each model exhibits distinct advantages in specific areas, reflecting their optimized and customized capabilities. In contrast, models with relatively shorter or limited Chain-of-Thought (CoT) reasoning, such as GPT-4.1, lag significantly behind in advanced data structures and algorithm tasks, suggesting they are predominantly effective only for conventional or simpler algorithmic problems. This indicates the importance of extended reasoning capabilities and fine-grained optimizations for achieving well-rounded, competitive algorithmic proficiency.

A.10 Participant Recruitment and Compensation

For the six Informatics Olympiad participants, they are all undergraduate students from Chinese universities with a computer science background, and have achieved at least a silver medal in the China Collegiate Programming Contest (CCPC) [3] or International Collegiate Programming Contest (ICPC) [4]. They work on a part-time basis. We will elaborate on the differences and connections between CCPC and ICPC in detail in the following sections.

We employ a total of five translators, including two native English speakers residing in the United States and three native Chinese speakers residing in China. Each question is either translated by a native English speaker and quality-checked by a native Chinese speaker, or translated by a native Chinese speaker and quality-checked by a native English speaker. They all hold at least a bachelor's degree in Master of Translation and Interpreting for English/Chinese translation and have a minimum of two years of professional experience in translation. They work full-time with a daily workload of less than 8 hours.

For crowd-sourced participants of human-comparable evaluation, we publish competitions on the Internet[5] and invite student contestants from various Chinese universities. Each competition offers subsidies to participants. Due to the high difficulty level of the competition problems, we require that contestants have prior experience in competitions at the ICPC level. To protect participants' privacy,

[3] `https://ccpc.io/`

[4] `https://icpc.global/`

[5] We will release this competition website link after the paper is accepted.

we do not collect any personal information other than verification of their competition experience, and all problem-solving records are collected anonymously.

It should be noted that this study does not focus on topics related to Human-Computer Interaction. Therefore, we do not ensure diversity in participant proficiency, ethnicity, gender, or other demographic characteristics, consistent with previous work comparing human and machine performance [6].

All the above personnel receive hourly wages (with crowd-sourced participants' subsidies calculated based on competition time) or daily wages in accordance with local labor regulations.

A.11 Details about ICPC and CCPC competitions

The International Collegiate Programming Contest (ICPC) is a prestigious, long-standing global algorithmic programming competition organized under the auspices of the Association for Computing Machinery (ACM). Targeting university-level students, the contest emphasizes the design and implementation of efficient algorithms to solve complex computational problems under time constraints. Teams consist of three students sharing a single computer, and they are evaluated based on the number of problems correctly solved and the total time taken, including penalties for incorrect submissions.

In contrast, the China Collegiate Programming Contest (CCPC) is a national-level competition established to foster programming education and competitive coding skills among Chinese university students. While CCPC is independently organized within China, it adopts a format and assessment mechanism closely aligned with that of ACM-ICPC, including the team structure, problem style, and evaluation metrics.

Although ACM-ICPC and CCPC share equivalent technical requirements and assessment criteria, they differ primarily in scope and recognition. ACM-ICPC is international in nature and culminates in a world final that attracts top teams from global universities, thereby serving as a benchmark for international excellence in algorithmic problem solving. CCPC, on the other hand, is confined to the domestic context of China, yet remains highly competitive and plays a significant role in the cultivation of algorithmic talent within the country.

Given the structural and technical equivalence between CCPC and ACM-ICPC — including problem format, evaluation criteria, and competitive rigor — these individuals can be considered ICPC-level in terms of algorithmic problem-solving proficiency, particularly within the context of Chinese competitive programming training systems.

Therefore, in the main text of this paper, we do not strictly distinguish between the two competitions. To avoid confusion, we uniformly use *ICPC* to refer to contests at this level.

In our benchmark, we evaluate each model's coding ability across four programming languages: C++, Python, Java, and JavaScript.

Programming Languages C++, Python, and Java are the official programming languages of the International Collegiate Programming Contest (ICPC) [6], and are widely adopted in competitive programming. Assessing these languages enables us to measure model performance in standard algorithmic and data structure tasks that are representative of real-world competitive programming scenarios.

In addition to these, we include JavaScript in our evaluation for several reasons. First, JavaScript is one of the most popular programming languages worldwide and is widely used in industry, particularly for web development and scripting tasks. Evaluating JavaScript allows us to assess the models' versatility and practical coding ability beyond traditional competitive programming settings. Furthermore, JavaScript's dynamic typing and unique language features introduce additional challenges for code generation and understanding, making it a valuable supplement for comprehensive evaluation of large language models.

By covering these four languages, our benchmark provides a broader and more realistic assessment of models' programming capabilities across both academic and industrial contexts.

[6] `https://icpc.global/regionals/rules`

	Confidence Interval		Computing Resources Usage	
	OIBench	OIBench Pseudo	GPU type	expected hours for all experiments
Base Models				
DeepSeek-V3-base	3.01	3.08	H800-80G * 16	4
Qwen2.5-Coder-32B-Base	1.14	2.89	A100-80G * 8	3
Qwen2.5-72B-base	2.01	2.91	A100-80G * 16	4
Llama-3.1-405B-base	2.01	2.91	A100-80G * 16	7
Instruction Tuned Non-Reasoning Models				
GPT-4.1🔒	1.75	2.52	API	-
DeepSeek-V3-0324	2.20	2.88	H800-80G * 16	8
Doubao-Seed-1.6-Off🔒	2.77	2.64	API	-
DeepSeek-V3-1226	1.01	2.57	H800-80G * 16	6
Claude3.5 Sonnet🔒	0.87	2.64	API	-
GPT-4O🔒	1.98	2.84	API	-
Qwen2.5-Coder-32B	0.92	2.58	A100-80G * 8	2
Llama-3.1-405B	0.48	2.28	A100-80G * 16	7
Reasoning Models				
Deepseek-R1-0528	1.74	2.75	H800-80G * 16	24
O4-mini-high🔒	1.80	2.49	API	-
O3-mini-high🔒	2.53	3.03	API	-
Doubao-Seed-1.6-On🔒	2.77	2.58	API	-
Qwen3-32B	1.24	2.53	H800-80G * 16	20
O1🔒	1.25	2.89	API	-
QwQ-32B	0.91	2.42	H800-80G * 16	18

Table 7: Confidence interval and computing resources usage for each model.

````
Please use {language} language to solve the following problem. Please use the OJ's input and output format:
```
{problem description and sample input/output}
```
````

Listing 1: Full problem prompt of OIBench for Instruction Models in English.

A.12 Prompts

We list the prompts used in the paper from Listing 1 to 8.

````
Please use {language} language to solve the following problem. Please use the OJ's input and output format:
```
{problem description and sample input/output}
```

```{language}
````

Listing 2: Full problem prompt of OIBench for Base Models in English. A markdown syntax is added to aid code generation.
````

````
Please use {language} language to solve the following problem. Please use the OJ's input and output format:

```
{problem description and sample input/output}
```

Pseudo code solution for the problem.
```
{Pseudocode}
```
````

Listing 3: Full problem prompt of OIBench Pseudo for Instruction Models in English.

````
Please use {language} language to solve the following problem. Please use the OJ's input and output format:

```
{problem description and sample input/output}
```

Pseudo code solution for the problem.
```
{Pseudocode}
```

```{language}
````

Listing 4: Full problem prompt of OIBench Pseudo for Base Models in English. A markdown syntax is added to aid code generation.

````
I'll give you a problem and its code, the code has some bugs, please fix the code according to the problem, and return the fixed code.

Task:
```md
{problem}
```


Code:
```cpp
{code}
```
````

Listing 5: Full problem prompt of Bugfix task.
````

````
I'll give you a problem and its code, the code has some missing parts, please complete the code according to the problem, and return the complete solution.

Task:
```md
{problem}
```

Code:
```cpp
{code}
```
````

Listing 6: Full problem prompt of Code Complete task.

````
please translate the following C++ code to {language} code

```cpp
{code}
```
````

Listing 7: Full problem prompt of Code Translate task.

````
Below is a C++ code and its corresponding input. Please guess its output based on the input. Please return the output using markdown syntax.
```cpp
{code}
```

```
{input_block}
```
````

Listing 8: Full problem prompt of Code Interprete task.

References

[1] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé 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, Joshua 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, and Wojciech Zaremba. Evaluating large language models trained on code. CoRR, abs/2107.03374, 2021.

[2] 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.

[3] Seyed-Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. CoRR, abs/2410.05229, 2024.

[4] Chulin Xie, Yangsibo Huang, Chiyuan Zhang, Da Yu, Xinyun Chen, Bill Yuchen Lin, Bo Li, Badih Ghazi, and Ravi Kumar. On memorization of large language models in logical reasoning. CoRR, abs/2410.23123, 2024.

[5] Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. CoRR, abs/2203.07814, 2022.

[6] Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, Zekun Wang, Jian Yang, Zeyu Cui, Yang Fan, Yichang Zhang, Binyuan Hui, and Junyang Lin. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. CoRR, abs/2501.01257, 2025.

[7] Quan Shi, Michael Tang, Karthik Narasimhan, and Shunyu Yao. Can language models solve olympiad programming? CoRR, abs/2404.10952, 2024.

[8] 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. CoRR, abs/2403.07974, 2024.

[9] Dong Huang, Yuhao Qing, Weiyi Shang, Heming Cui, and Jie Zhang. Effibench: Benchmarking the efficiency of automatically generated code. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024.

[10] Tristan Coignion, Clément Quinton, and Romain Rouvoy. A performance study of llm-generated code on leetcode. In Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering, EASE 2024, Salerno, Italy, June 18-21, 2024, pages 79–89. ACM, 2024.

[11] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025.

[12] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. arXiv preprint arXiv:2412.16720, 2024.

[13] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837, 2022.

[14] Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. In Findings of the Association for Computational Linguistics: ACL 2023, pages 1049–1065, 2023.

[15] Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, et al. Towards large reasoning models: A survey of reinforced reasoning with large language models. arXiv preprint arXiv:2501.09686, 2025.

[16] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. CoRR, abs/2110.14168, 2021.

[17] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In Joaquin Vanschoren and Sai-Kit Yeung, editors, Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual, 2021.

[18] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018, pages 2369–2380. Association for Computational Linguistics, 2018.

[19] Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. Logicbench: Towards systematic evaluation of logical reasoning ability of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024, pages 13679–13707. Association for Computational Linguistics, 2024.

[20] Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. Criticbench: Benchmarking llms for critique-correct reasoning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024, pages 1552–1587. Association for Computational Linguistics, 2024.

[21] Shubham Parashar, Blake Olson, Sambhav Khurana, Eric Li, Hongyi Ling, James Caverlee, and Shuiwang Ji. Inference-time computations for LLM reasoning and planning: A benchmark and insights. CoRR, abs/2502.12521, 2025.

[22] Yijia Xiao, Edward Sun, Tianyu Liu, and Wei Wang. Logicvista: Multimodal LLM logical reasoning benchmark in visual contexts. CoRR, abs/2407.04973, 2024.

[23] Bahare Fatemi, Mehran Kazemi, Anton Tsitsulin, Karishma Malkan, Jinyeong Yim, John Palowitch, Sungyong Seo, Jonathan Halcrow, and Bryan Perozzi. Test of time: A benchmark for evaluating llms on temporal reasoning. CoRR, abs/2406.09170, 2024.

[24] Jiayi Gui, Yiming Liu, Jiale Cheng, Xiaotao Gu, Xiao Liu, Hongning Wang, Yuxiao Dong, Jie Tang, and Minlie Huang. Logicgame: Benchmarking rule-based reasoning abilities of large language models. CoRR, abs/2408.15778, 2024.

[25] Xiao Liu, Zirui Wu, Xueqing Wu, Pan Lu, Kai-Wei Chang, and Yansong Feng. Are llms capable of data-based statistical and causal reasoning? benchmarking advanced quantitative reasoning with data. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024, pages 9215–9235. Association for Computational Linguistics, 2024.

[26] Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, and Tao Xie. Codereval: A benchmark of pragmatic code generation with generative pre-trained models. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024, pages 37:1–37:12. ACM, 2024.

[27] Jiasheng Zheng, Boxi Cao, Zhengzhao Ma, Ruotong Pan, Hongyu Lin, Yaojie Lu, Xianpei Han, and Le Sun. Beyond correctness: Benchmarking multi-dimensional code generation for large language models. CoRR, abs/2407.11470, 2024.

[28] Yihong Dong, Jiazheng Ding, Xue Jiang, Zhuo Li, Ge Li, and Zhi Jin. Codescore: Evaluating code generation by learning code execution. CoRR, abs/2301.09043, 2023.

[29] Alex Gu, Baptiste Rozière, Hugh James Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net, 2024.

[30] Anjiang Wei, Jiannan Cao, Ran Li, Hongyu Chen, Yuhui Zhang, Ziheng Wang, Yaofeng Sun, Yuan Liu, Thiago S. F. X. Teixeira, Diyi Yang, Ke Wang, and Alex Aiken. Equibench: Benchmarking code reasoning capabilities of large language models via equivalence checking. CoRR, abs/2502.12466, 2025.

[31] Jiarong Wu, Songqiang Chen, Jialun Cao, Hau Ching Lo, and Shing-Chi Cheung. Isolating language-coding from problem-solving: Benchmarking llms with pseudoeval. CoRR, abs/2502.19149, 2025.

[32] Batu Guan, Xiao Wu, Yuanyuan Yuan, and Shaohua Li. Is your benchmark (still) useful? dynamic benchmarking for code language models. CoRR, abs/2503.06643, 2025.

[33] Simin Chen, Pranav Pusarla, and Baishakhi Ray. Dynamic benchmarking of reasoning capabilities in code large language models under data contamination. CoRR, abs/2503.04149, 2025.

[34] King Zhu, Qianbo Zang, Shian Jia, Siwei Wu, Feiteng Fang, Yizhi Li, Shawn Gavin, Tuney Zheng, Jiawei Guo, Bo Li, et al. Lime: Less is more for mllm evaluation. arXiv preprint arXiv:2409.06851, 2024.

[35] Shangqing Tu, Kejian Zhu, Yushi Bai, Zijun Yao, Lei Hou, and Juanzi Li. DICE: detecting in-distribution contamination in llm's fine-tuning phase for math reasoning. CoRR, abs/2406.04197, 2024.

[36] Tianwen Wei, Liang Zhao, Lichang Zhang, Bo Zhu, Lijie Wang, Haihua Yang, Biye Li, Cheng Cheng, Weiwei Lü, Rui Hu, Chenxia Li, Liu Yang, Xilin Luo, Xuejie Wu, Lunan Liu, Wenjun Cheng, Peng Cheng, Jianhao Zhang, Xiaoyu Zhang, Lei Lin, Xiaokun Wang, Yutuan Ma, Chuanhai Dong, Yanqi Sun, Yifu Chen, Yongyi Peng, Xiaojuan Liang, Shuicheng Yan, Han Fang, and Yahui Zhou. Skywork: A more open bilingual foundation model. CoRR, abs/2310.19341, 2023.

[37] Ruijie Xu, Zengzhi Wang, Run-Ze Fan, and Pengfei Liu. Benchmarking benchmark leakage in large language models. CoRR, abs/2404.18824, 2024.

[38] Chunyuan Deng, Yilun Zhao, Xiangru Tang, Mark Gerstein, and Arman Cohan. Investigating data contamination in modern benchmarks for large language models. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard, editors, Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language

Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024, pages 8706–8719. Association for Computational Linguistics, 2024.

[39] Cheng Xu, Shuhao Guan, Derek Greene, and M. Tahar Kechadi. Benchmark data contamination of large language models: A survey. CoRR, abs/2406.04244, 2024.

[40] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. Linux J., 2014(239), March 2014.

[41] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, and S. S. Li. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. CoRR, abs/2501.12948, 2025.

[42] Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations. arXiv preprint arXiv:2411.00640, 2024.

[43] Francisco Eiras, Aleksandar Petrov, Bertie Vidgen, Christian Schröder de Witt, Fabio Pizzati, Katherine Elkins, Supratik Mukhopadhyay, Adel Bibi, Botos Csaba, Fabro Steibel, Fazl Barez, Genevieve Smith, Gianluca Guadagni, Jon Chun, Jordi Cabot, Joseph Marvin Imperial, Juan A. Nolazco-Flores, Lori Landay, Matthew Thomas Jackson, Paul Röttger, Philip H. S. Torr, Trevor Darrell, Yong Suk Lee, and Jakob N. Foerster. Position: Near to mid-term risks and opportunities of open-source generative AI. In Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net, 2024.

[44] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437, 2024.

[45] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. arXiv preprint arXiv:2409.12186, 2024.

[46] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. arXiv preprint arXiv:2412.15115, 2024.

[47] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.

[48] OpenAI. Gpt-4.1 model documentation. `https://platform.openai.com/docs/models/gpt-4.1`, 2025.

[49] Seed. Seed:introduction to techniques used in seed1.6. `https://seed.bytedance.com/en/seed1_6`, 2025.

[50] Claude. claude 3.5 sonnet. `https://www.anthropic.com/news/claude-3-5-sonnet`, 2025. Accessed: 2025-05-10.

[51] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. arXiv preprint arXiv:2410.21276, 2024.

[52] OpenAI. O4-mini. https://openai.com/index/introducing-o3-and-o4-mini/, 2025. Accessed: 2025-05-10.

[53] OpenAI. O3-mini. https://openai.com/index/openai-o3-mini/, 2025. Accessed: 2025-05-10.

[54] Qwen. Qwq-32b: Embracing the power of reinforcement learning, March 2025.

[55] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. CoRR, abs/2408.03314, 2024.

[56] Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I Wang. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. arXiv preprint arXiv:2502.18449, 2025.

[57] Chao Lei, Yanchuan Chang, Nir Lipovetzky, and Krista A. Ehinger. Planning-driven programming: A large language model programming workflow. CoRR, abs/2411.14503, 2024.

[58] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. arXiv preprint arXiv:1909.08053, 2019.

[59] Shukai Liu, Linzheng Chai, Jian Yang, Jiajun Shi, He Zhu, Liran Wang, Ke Jin, Wei Zhang, Hualei Zhu, Shuyue Guo, Tao Sun, Jiaheng Liu, Yunlong Duan, Yu Hao, Liqun Yang, Guanglin Niu, Ge Zhang, and Zhoujun Li. Mdeval: Massively multilingual code debugging. CoRR, abs/2411.02310, 2024.

[60] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation. In Joaquin Vanschoren and Sai-Kit Yeung, editors, Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual, 2021.

[61] Weixiang Yan, Yuchen Tian, Yunzhe Li, Qian Chen, and Wen Wang. Codetransocean: A comprehensive multilingual benchmark for code translation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023, pages 5067–5089. Association for Computational Linguistics, 2023.

[62] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles, 2023.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The paper propose a new coding benchmark for LLMs, namely OIBench, with various extensive experiments, which is reflected in the abstract and introduction.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitation in the appendix.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: [NA]

Guidelines: There are no theoretical results in the paper.

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide the experiment enviroments in Section 4 Evaluation Environment paragraph.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We open-source first 100 problems in `https://huggingface.co/datasets/meituan/OIBench`, and will fully open-source 250 problems after paper acceptance.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer:[Yes]

   Justification: For training in anti-contamination experiments, we report the details in Section A.4 . For testing, we provide the experiment enviroments in Section 4. paragraph.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer:[Yes]

   Justification: We report in statistical significance Section A.6 using the method suggested by [42]. The performance gap between model tiers significantly exceeds their confidence intervals.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: For training in anti-contamination experiments, we report the details in Section A.4. For testing, we report the details in Table 7.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: : The data is constructed under NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: As a coding benchmark, OIBench has no societal impacts as we might expect.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: As a coding benchmark, OIBench has no such risks as we might expect.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We credit all models used in the paper via citation.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The assets is included in the HuggingFace Repo of OIBench.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

    Justification: The crowdsourcing invites human participants to clean or do OIBench problems in order to get human-model comparison results. We provide human participants with detailed task instructions, inform them of data open-source plans, and compensate them in accordance with local labor regulations as part-time workers, as shown in Appendix A.10.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The crowdsourcing experiments do not study human as subjects in the paper.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: We use LLM only for writing, editing, or formatting purposes.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.