Humanity's Last Code Exam: Can Advanced LLMs Conquer Human's Hardest Code Competition?

Xiangyang Li* **Xiaopeng Li*** **Kuicai Dong** **Quanhu Zhang**
Rongju Ruan **Xinyi Dai†** **Xiaoshuang Liu** **Shengchun Xu**
Yasheng Wang **Ruiming Tang†**
HUAWEI NOAH'S ARK LAB
{lixiangyang34, dong.kuicai, zhangquanhu1, ruanrongju, daixinyi5,
liuxiaoshuang4, xushengchun, wangyasheng, tangruiming}@huawei.com
xiaopli2-c@my.cityu.edu.hk

Abstract

Code generation is a core capability of large language models (LLMs), yet mainstream benchmarks (e.g., APPs and LiveCodeBench) contain questions with medium-level difficulty and pose no challenge to advanced LLMs. To better reflected the advanced reasoning and code generation ability, We introduce **H**umanity's **L**ast **C**ode **E**xam (**HLCE**), comprising 235 most challenging problems from the International Collegiate Programming Contest (ICPC World Finals) and the International Olympiad in Informatics (IOI) spanning 2010 – 2024. As part of HLCE, we design a harmonized online–offline sandbox that guarantees fully reproducible evaluation. Through our comprehensive evaluation, we observe that even the strongest reasoning LLMs: o4-mini(high) and Gemini-2.5 Pro, achieve pass@1 rates of only 15.9% and 11.4%, respectively. Meanwhile, we propose a novel "self-recognition" task to measure LLMs' awareness of their own capabilities. Results indicate that LLMs' self-recognition abilities are not proportionally correlated with their code generation performance. Finally, our empirical validation of test-time scaling laws reveals that current advanced LLMs have substantial room for improvement on complex programming tasks. We expect HLCE to become a milestone challenge for code generation and to catalyze advances in high-performance reasoning and human–AI collaborative programming. Our code and dataset are also public available[1].

1 Introduction

Large Language Models (LLMs) (Liu et al., 2024; Achiam et al., 2023; Jaech et al., 2024; Hui et al., 2024) have demonstrated human-level proficiency across a wide range of text understanding (Dong et al., 2023), reasoning (Yang et al., 2025b), and generation tasks. Among these, code understanding and generation (Hui et al., 2024; Li et al., 2025; Guo et al., 2024; Zhu et al., 2024) have emerged as key research areas, since code inherently reflects reasoning and logical thinking skills. To evaluate LLMs' capabilities in this domain, many code generation benchmarks, such as LiveCodeBench (Jain et al., 2024), MBPP (Austin et al., 2021), HumanEval (Chen et al., 2021), and SWE-bench (Jimenez et al., 2023), have been established. These benchmarks have accelerated the progress of LLM-based code models, but the most advanced LLMs are now achieving near-perfect or saturated performance on many of these tasks. This raises the question: Do current benchmarks truly reflect the advanced reasoning and code generation abilities of state-of-the-art LLMs, especially when it comes to complex, structured, or algorithmic coding challenges?

Current benchmarks suffer from several critical limitations: **(1) Limited Difficulty:** With the increasing capabilities of LLMs, many benchmarks have become too easy, as shown in Figure 1. For example, benchmarks like HumanEval already report high pass rates, and achieving perfect scores is now a matter of incremental progress. **(2) Absence of Interactive-based Evaluation:** Most contemporary code competition benchmarks (e.g., LiveCodeBench) rely on standard I/O judging, where code submissions are evaluated via input/output pairs. This facilitates outcome-reward reinforcement learning (Guo et al., 2025; Yang et al., 2025a; Duong et al., 2025), allowing models to exploit these feedback loops. However, interactive-based judging[2] remains largely unexplored for LLMs. The abilities of LLMs in this setting are thus

*These authors contributed equally to this work.
†Corresponding authors.
[1] https://github.com/Humanity-s-Last-Code-Exam/HLCE

[2] Interactive-based judging is such as International Olympiad in Informatics (IOI) problems, where participants implement specific function signatures and the evaluation system interacts with these functions directly

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 21122–21137
November 4-9, 2025

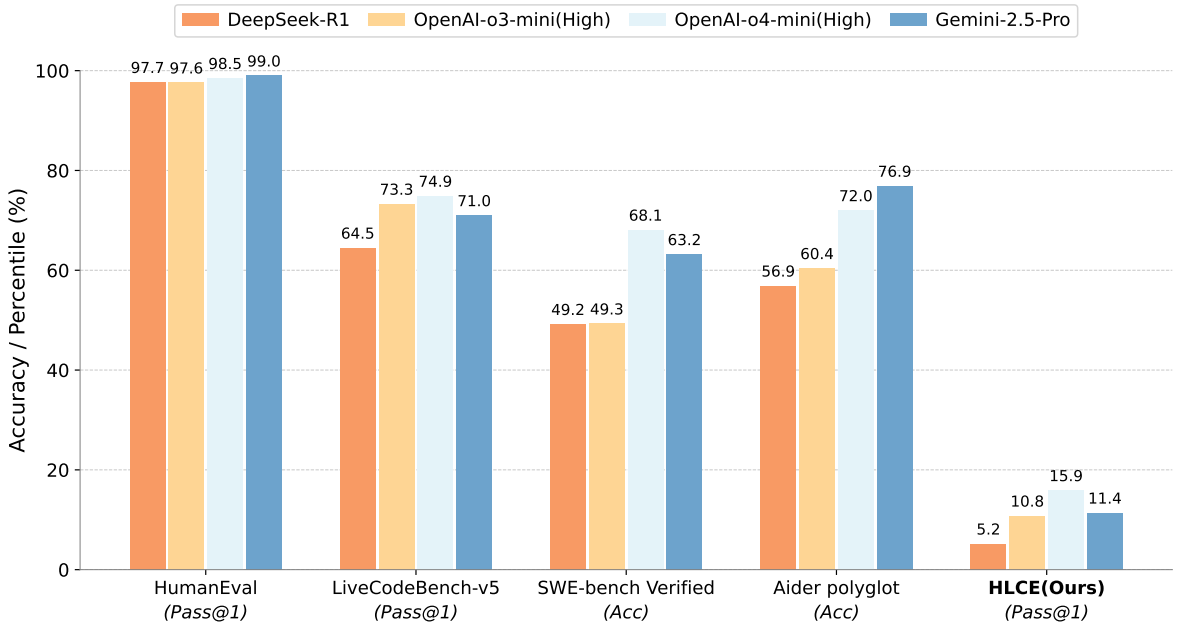


Figure 1: Performance of advanced LLMs on widely-used code generation benchmarks and HLCE.

insufficiently characterized. **(3) Underexplored Test-time Scaling Laws:** The relationship between model scale and performance at test-time in code generation is insufficiently probed. While recent models like o4-mini and Gemini-2.5-pro have achieved impressive results, the scalability and potential upper bounds of test-time performance are still open questions that current benchmarks do not address.

To address these limitations, we introduce a new benchmark, Humanity's Last Code Exam (HLCE), comprising carefully curated and rigorously filtered programming problems sourced from the IOI and ICPC World Finals competitions from 2010 to 2024. After extensive manual cleaning and validation, the benchmark contains 235 high-quality competitive programming problems, each accompanied by comprehensive test cases. HLCE extends beyond traditional ACM input/output formats to include interactive challenges that demand dynamic program behavior and real-time interaction.

Using HLCE, we evaluate 12 leading LLMs, including both reasoning and non-reasoning models. Our results indicate that even state-of-the-art models o4-mini(high) and Gemini-2.5 Pro achieve only 15.1% and 11.4% pass@1 rates, respectively, on HLCE. This demonstrates that HLCE presents a significantly higher level of difficulty compared to existing code generation benchmarks. Furthermore, we introduce a novel self-recognition task, in which LLMs must determine whether their generated code solutions are correct or incorrect, providing direct insight into the models' capacity for self-assessment and error recognition. Additionally, HLCE enables empirical analysis of test-time scaling laws. Our findings reveal that, despite the impressive capabilities of models like o4-mini(high) and Gemini-2.5 Pro, the upper bounds of test-time performance remain largely unexplored, suggesting considerable room for LLM-based reasoning.

Finally, we consider an intriguing question: Can advanced LLMs genuinely compete with top human programmers in IOI and ICPC World Finals? To explore this, we collected performance data for medalists from these competitions spanning 2010 to 2024 and directly compared their results to those of leading LLMs. Our analysis shows that models such as o3-mini, Gemini-2.5-pro, and o4-mini are capable of achieving medal-level performance in ICPC competitions. However, these models still underperform compared to human medalists in the IOI, highlighting the persisting challenges in truly top-tier human-competitive code generation. In summary, our contributions are as follows:

- We introduce HLCE, a novel benchmark comprising 235 competitive programming problems from IOI and ICPC World Finals (2010-2024), featuring both standard and interactive programming challenges that significantly exceed the difficulty of existing code generation benchmarks.

- We conduct comprehensive evaluations on 12 leading LLMs, showing that even most advanced LLMs achieve only 15.1% and 11.4% pass@1

Benchmark	Diff' level	Language	Test Cases	Problem Category	Human Results
OlympicArena	▰▱	-	✗	Interactive	✗
CodeContest	▰▱	C++	✓	Standard I/O	✗
LiveCodeBench	▰▱	Python	✓	Standard I/O	✗
APPS	▰▱	Python	✓	Standard I/O	✗
HLCE (Ours)	▰	C++, Python	✓	Standard I/O &Interactive	✓

Table 1: HLCE vs other code competition benchmarks.

rates. We also propose a novel self-recognition task to measure models' abilities to recognize the correctness of their own generated solutions.

- We empirically validate test-time scaling laws on HLCE, demonstrating that current LLMs have not yet reached their performance upper bounds and highlighting substantial room for further advancement via improved reasoning capabilities.
- We provide comparative analyses with top human competitors, revealing the gap between advanced LLMs and competition medalists.

2 Related Work

Large Language Models for Code Generation. Recent advancements in LLMs have significantly enhanced the capability of automated code generation. Models such as Codex (Chen et al., 2021), StarCoder (Li et al., 2023), and CodeLlama (Roziere et al., 2023) have demonstrated remarkable proficiency in understanding and generating programming code across various languages. The emergence of instruction-tuned models like ChatGPT (Achiam et al., 2023) and Claude (Anthropic, 2025) has further pushed the boundaries of code generation capabilities, allowing for more contextually appropriate and functionally correct code outputs. More recently, reasoning-enhanced models have made substantial progress in the code generation domain, with claude-3.7 (Anthropic, 2025), deepSeek-r1 (Guo et al., 2025), and o4-mini (OpenAI, 2025) exhibiting extraordinary capabilities in producing complex, functional code.

Code Generation Benchmarks. LLM code generation evaluation has evolved through diverse benchmarks. HumanEval (Chen et al., 2021) established a standard with Python function completion tasks, while MBPP (Austin et al., 2021) expanded this approach with varied difficulty problems. Specialized benchmarks emerged targeting different programming aspects: CodeContests (Li et al., 2022) and APPS (Hendrycks et al., 2021) focus on competitive programming challenges, while DS-1000 (Lai et al., 2023) addresses data science tasks. Recent developments include LiveCodeBench (Jain et al., 2024) for measuring performance on coding competitions across different time periods, Aider (Aider, 2024) evaluate LLM's ability to follow instructions and edit code successfully without human intervention, and SWE-Bench (Jimenez et al., 2023) for real-world GitHub issues. Open-R1 (Face, 2025) maintained a leaderboard of IOI 2024, but only a few problems were prone to causing errors and overfitting. As illustrated in Figure 1, HLCE distinguishes itself through its exceptional difficulty level: even the most advanced reasoning-focused LLMs struggle to perform well, thereby establishing a new ceiling for evaluating the code generation capabilities of current models.

3 HLCE Benchmark

As illustrated in Figure 2, the HLCE framework comprises three principal components: **Dataset**, **Evaluation Task**, and **Evaluation Framework**.

3.1 Dataset

We collect our dataset from two premier competitive programming contests: the International Olympiad in Informatics (IOI) and the International Collegiate Programming Contest (ICPC) World Finals. The competitions represent state-of-the-art algorithmic problem-solving challenges, attracting elite participants worldwide and featuring meticulously designed problems that evaluate advanced computational thinking capabilities.

For **ICPC World Finals**, our dataset construction process is as follows:

- **Extracting Problem Statements**: Since official problem descriptions are only available in PDF format, we first attempted to use various PDF parsing tools (e.g., Markitdown (Microsoft, 2024)), but none yielded satisfactory results. Hence, we manually copied and pasted problem content from the PDFs, then used ChatGPT to further refine and standardize the formatting.
- **Collecting Test Cases**: We carefully gathered all official test cases for each problem to ensure completeness and accuracy for evaluation.
- **Data Filtering**: We removed problems with corrupted or missing official test cases (such as some problems from 2018), as well as interactive prob-

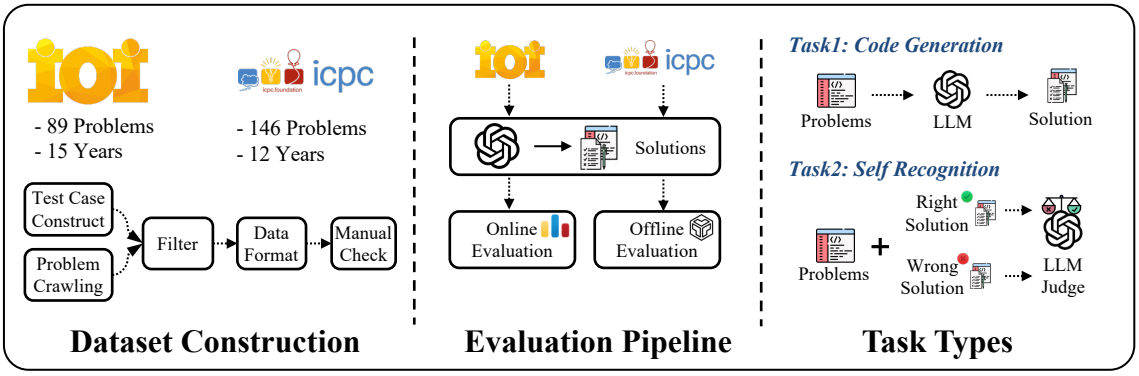


Figure 2: HLCE Benchmark Overview.

lems incompatible with the standard ACM input/output format.

This rigorous process resulted in a dataset of 146 problems from the 2011–2023 ICPC World Finals, each accompanied by complete test cases.

For **IOI**, the construction process is as follows:

- **Extracting Problem Statements**: We identified a comprehensive dump[3] of IOI problems on Codeforces and systematically extracted all problem descriptions using web scraping techniques. The raw extracted content was cleaned and reformatted using ChatGPT to resolve formatting and consistency issues.
- **Collecting Test Cases**: We gathered all official test cases corresponding to each problem for evaluation purposes.
- **Data Filtering**: Output-only problems, which require only output files instead of program code, were excluded due to evaluation challenges.

The resulting IOI dataset includes 89 problems from 2010–2024, each with full test cases and standardized formatting.

3.2 Evaluation Tasks

We define two distinct evaluation tasks in our benchmark. The first is the classic **code generation** task, which serves as a standard metric to evaluate the ability of LLMs to generate accurate code. In addition to this, we propose a novel task termed the **self-recognition** task, which aims to evaluate the model's ability to recognize whether the code it generates is correct or incorrect. This task provides a unique perspective on the model's reasoning and introspection capabilities, which are crucial for applications requiring reliable and autonomous coding solutions.

[3]https://ioi.contest.codeforces.com/group/32KGsXgiKA/blog

3.3 Evaluation Framework

For the code generation task on ICPC World Finals problems, we implemented a Python-based evaluation framework that utilizes standard test cases with predefined input/output pairs.

For the IOI dataset, despite having collected all test cases, most IOI problems require interaction between the evaluation system and C++ functions in the submission, necessitating the implementation of specific evaluation programs for each problem. To address this issue, we developed an automated submission bot that interfaces with the Codeforces IOI platform. This bot submits LLM-generated solutions to the official Codeforces IOI judging system and retrieves detailed scoring information and execution results for each problem.

For the self-recognition task, we implemented a separate Python evaluation framework designed to assess LLMs' ability to accurately recognize their own capabilities.

4 Experiments

In this section, we evaluate and analyze the performance of current state-of-the-art LLMs on two tasks from the HLCE benchmark.

4.1 Experimental Setup

Benchmarked Models. To conduct a comprehensive evaluation, we assessed a diverse set of SOTA LLMs, encompassing both reasoning and non-reasoning models. For non-reasoning LLMs, we selected **gpt-4o-mini**, **claude-3-5-sonnet-20241022**, **claude-3-7-sonnet-20250219**, **gpt-4o-2024-05-13**, **deepseek-v3-0324**, and **chatgpt-4o-latest**. The reasoning models included **o1-mini**, **o1**, **o3-mini (High)**, **Gemini-2.5-Pro**, **DeepSeek-R1**, and **claude-3.7-sonnet-thinking**.

Implementation Details. For all models, we

utilized the OpenRouter[4] API with default hyperparameters. We generated 5 responses per question and evaluated performance using pass@1 (Chen et al., 2021) and pass@5 metrics. For IOI questions, scoring is based on test case success, with 100 indicating all tests passed. Scores below 100 are considered failures. Additional details are provided in Appendix A.

4.2 Evaluation Results on HLCE

4.2.1 Results on Code Generation Task

Based on the results in Table 2, we can observe several significant trends in the performance of LLMs on the HLCE code generation task:

Task Difficulty The results highlight the extraordinary difficulty of our HLCE benchmark compared to existing code generation tasks. While o4-mini(high) achieved impressive scores on standard benchmarks (98.5% pass@1 on HumanEval, 74.9% pass@1 on LivecodeBench, 68.1% pass@1 on SWE-bench Verified, and 72.0% on Aider), it only achieved a pass@1 rate of 15.85% on HLCE. This significant performance gap reveals several issues: HLCE problems are exceptionally challenging, particularly those from IOI (6.48% pass@1 rate), requiring advanced computational thinking that current Large Language Models (LLMs) struggle to master; the gap between pass@1 and pass@5 rates (15.85% vs. 29.31%) indicates that models possess the necessary knowledge but lack reasoning consistency. This difficulty establishes HLCE as a critical benchmark for measuring advanced reasoning and algorithmic problem-solving capabilities in LLMs.

Non-reasoning vs. Reasoning Models A significant performance gap exists between non-reasoning and reasoning-enhanced LLMs. Almost all reasoning models consistently outperform non-reasoning counterparts across all metrics and datasets. The best reasoning model (o4-mini(high)) achieves 15.85% average pass@1, approximately 4.5 times higher than the best non-reasoning model (deepseek-v3-0324) at 3.53%. Notably, deepseek-v3-0324 exhibits superior performance among non-reasoning models, even surpassing the reasoning model Claude-3.7-Thinking. We attribute this capability to DeepSeek-V3-0324 being distilled from DeepSeek-R1 data, which enhanced its coding abilities. This suggests that targeted distillation from reasoning-rich data could yield models with advanced coding capabilities without requiring explicit reasoning during inference.

Model degradation Phenomenon An interesting exception to the general trend is the claude-3.7-thinking model, which underperforms compared to non-reasoning models and shows notably weaker results on IOI problems than both claude-3.5-sonnet and claude-3.7-sonnet. This model achieves 0% pass rates on IOI problems, representing a significant decline despite its enhanced reasoning capabilities. We hypothesize this degradation stems from Anthropic's optimization focus on general software engineering rather than competitive programming. The claude-3.7 technical report (Anthropic, 2025) indicates that Claude 3.7 Sonnet achieves 62.3% accuracy on SWE-bench Verified, outperforming o3-mini(high) (49.3%), suggesting a deliberate trade-off favoring practical software engineering tasks.

Standard I/O vs. Interactive Evaluation A striking observation is the significant performance gap of models between IOI and ICPC World Finals competitions. Even o4-mini(high), which achieves a pass@1 rate of 25.21% on ICPC World Finals, only manages 6.48% on IOI. We hypothesize that this substantial discrepancy stems from the training methodology of current reasoning models, which predominantly utilize Standard I/O-based data with outcome-reward RL. This approach aligns well with most ICPC problems but fails to address the interactive nature of IOI problems, which often require program interaction with a judge. This finding underscores the importance of incorporating more diverse training data and developing more robust RL training environments to enhance model performance across different problem types.

4.2.2 Results on Self-recognition Task

The self-recognition task evaluates models' ability to accurately judge the correctness of their own solutions, providing insights into their metacognitive capabilities. Table 3 presents the AUC scores for this task across various models.

Performance Comparison Among non-reasoning models, ChatGPT-4o-latest demonstrates superior self-recognition with an AUC of 0.84, despite showing only moderate performance in code generation (pass@1: 1.18%, pass@5: 2.37%). This suggests that strong self-recognition

[4]https://openrouter.ai/

Model	ICPC World Finals		IOI		Avg.	
	pass@1	pass@5	pass@1	pass@5	pass@1	pass@5
Non-reasoning						
gpt-4o-mini	0.96	2.99	0.00	0.00	0.48	1.50
claude-3.5-sonnet	2.74	5.04	0.67	1.12	1.71	3.08
claude-3.7-sonnet	3.84	6.41	**1.12**	1.12	2.48	3.77
gpt-4o-2024-05-13	1.99	3.35	0.45	1.12	1.22	2.24
deepseek-v3-0324	**6.16**	**12.1**	0.90	1.12	**3.53**	**6.61**
chatgpt-4o-latest	1.91	3.61	0.45	1.12	1.18	2.37
Reasoning						
claude-3.7-thinking	4.25	8.22	0.00	0.00	2.13	4.11
o1-mini	10.55	19.86	2.34	3.37	6.45	11.62
DeepSeek-R1	8.08	14.38	2.23	5.62	5.16	10.00
o3-mini (High)	13.42	29.45	**8.26**	10.23	10.84	19.84
Gemini-2.5-Pro	17.40	29.45	5.39	11.24	11.40	20.35
o4-mini (High)	**25.21**	**43.84**	6.48	**14.77**	**15.85**	**29.31**

Table 2: Performance results on HLCE code generation task.

	Model Name	AUC↑
Non-reasoning	gpt-4o-mini	0.60
	claude-3.5-sonnet	0.75
	claude-3.7-sonnet	0.69
	gpt-4o-2024-05-13	0.76
	deepseek-v3-0324	0.63
	chatgpt-4o-latest	**0.84**
Reasoning	claude-3.7-thinking	0.79
	o1-mini	0.73
	DeepSeek-R1	**0.81**
	o3-mini (High)	0.66
	Gemini-2.5-Pro	0.72
	o4-mini (High)	0.63

Table 3: Performance results on self-recognition task, ↑ denotes higher is better.

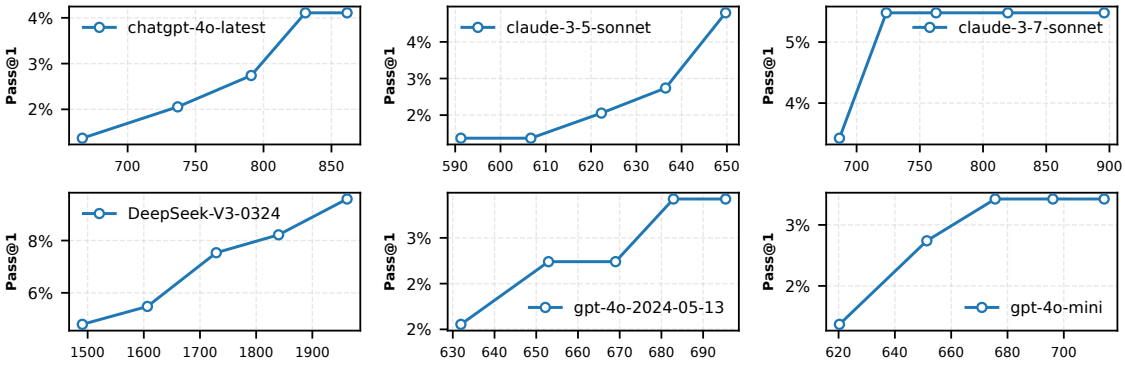
capability does not necessarily correlate with superior code generation performance. Similarly, DeepSeek-R1 achieves the highest AUC (0.81) among reasoning models while ranking in the middle tier for code generation.

Knowledge of Self-Knowledge: The Socratic Paradox in LLMs The Socratic paradox, encapsulated in the statement "I know that I know nothing," represents the philosophical understanding that true wisdom begins with recognizing the limits of one's knowledge. However, across the two tasks in HLCE, we observe a contrasting phenomenon. Interestingly, the top-performing model in self-recognition comes from the non-reasoning category (**chatgpt-4o-latest: 0.84**), outperforming all reasoning models. This contrasts with the code generation task, where reasoning models (particularly o4-mini) significantly outperformed non-reasoning counterparts. This empirical evidence reveals a fundamental challeng: some models excel at problem-solving but lack accurate self-recognition, while others show better self-recognition despite lower performance. This disconnect suggests reasoning abilities and self-recognition develop along different trajectories in current LLM architectures, highlighting the need for research that enhances both dimensions simultaneously.

4.3 Test Time Scaling Law on HLCE

Test time scaling law, where increased computational resources during inference improve model performance (Jaech et al., 2024; Face, 2025; OpenAI, 2025), has been validated on Olympic-level mathematics (Muennighoff et al., 2025) but rarely on Olympic-level programming challenges, except by OpenAI (El-Kishky et al., 2025). Therefore, in this section, we utilize the extremely difficult HLCE benchmark to verify that the test time scaling law still holds for current SOTA LLMs. Specifically, we group all generated responses by token count and calculate the pass@1 rate for each group in ICPC World Finals. The results are presented in Figures 3a and 3b.

(a) Pass@1 vs. Average Output Tokens in Non-reasoning Models



(b) Pass@1 vs. Average Output Tokens in Reasoning Models

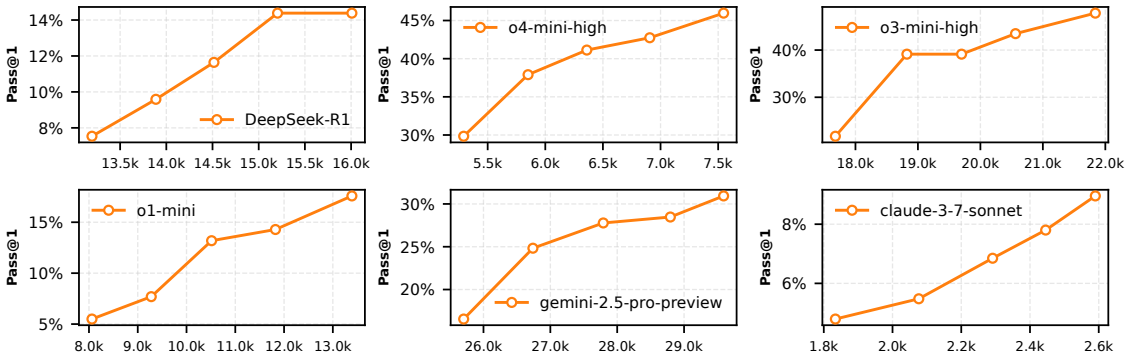


Figure 3: Comparison of Non-Reasoning and Reasoning Models in Test-time Scaling.

Test Time Scaling Law Holds As shown in the figures, both models with and without reasoning capabilities demonstrate a clear test time scaling law on the HLCE benchmark. Pass@1 rate gradually increases as thinking time increases. This indicates that complex programming problems benefit significantly from extended reasoning processes, and models can leverage additional computation time to refine their solutions and correct potential errors.

Test Time Scaling Law has not yet reached its Boundary Even for the most capable models such as o4-mini(high) and Gemini-2.5-pro, the pass@1 rate continues to increase with longer reasoning sequences. This suggests that despite the impressive capabilities of current state-of-the-art models, the test time scaling law **has not yet reached its limit**. Surprisingly, we observed that compared to o3-mini(high), o4-mini(high) reduced the average output token length by approximately two-thirds. This finding has important implications for the future development of code LLMs, indicating that we can confidently continue to optimize test-time scaling laws to further enhance model performance.

4.4 Can LLMs Surpass Top-tier Human?

To assess whether the latest LLMs can truly compete with elite human programmers, we compared the performance of SOTA LLMs against the gold, silver, and bronze medal thresholds from historical IOI and ICPC World Finals competitions. The comparative results are presented in Tables 4 and 5.

LLMs Could Reach Medal Level From Tables 4 and 5, we observe a remarkable finding: current sota models can achieve medal-winning performance in prestigious programming competitions such as IOI and ICPC World Finals. Notably, Gemini-2.5-pro and o4-mini(high) demonstrate exceptional capabilities, securing silver and gold medals in IOI and ICPC World Finals, respectively. This indicates that contemporary SOTA LLMs have developed computational reasoning capabilities that rival those of top human competitors. Furthermore, Gemini-2.5-Pro exhibits impressive consistency in IOI performance, earning five bronze medals and one silver medal. These results suggest that in the near future, we may reasonably anticipate LLMs achieving gold medal performance levels across such competitions.

IOI(Max Points)						
Year	Bronze	Silver	Gold	o3-mini (high)	Gemini 2.5 pro	o4-mini (high)
2024	217	289	360	133	138	89
2023	153	230	334	53	157	52
2022	147	258	416	5	150	138
2021	203	289	373	220	292	219
2020	236	338	480	92	237	119
2016	240	328	416	212	317	12
2015	185	326	440	211	119	162
2014	223	323	449	38	224	150
2013	220	359	480	100	203	323
2012	157	237	364	40	224	200
2011	267	370	478	0	224	0

Table 4: Human Performance thresholds for medals and the best LLM Performance in IOI. We utilize the highest score from the five responses.

ICPC World Finals(Solved Problems)						
Year	Bronze	Silver	Gold	o3-mini (high)	Gemini 2.5 pro	o4-mini (high)
2023	7	8	9	3	3	3
2022	8	8	9	2	3	4
2021	8	8	9	5	4	5
2020	9	10	11	2	4	6
2019	7	7	8	1	2	2
2018	7	7	8	0	1	2
2017	8	8	10	4	3	4
2016	9	9	10	5	4	6
2015	9	10	10	9	7	8
2014	4	5	6	2	4	4
2013	6	7	8	3	1	6
2012	6	7	7	4	3	8
2011	7	7	8	3	4	6

Table 5: Human Performance thresholds for medals and the best LLM Performance in ICPC World Finals. A problem is considered solved if any of the five responses solves it.

Discrepancy Between Medal Achievement and Pass@1 Rate The intriguing discrepancy observed in Table 4 and Table 2, where models like o4-mini and Gemini-2.5-Pro can achieve medal-worthy scores in IOI competitions while maintaining relatively low pass@1 rates, highlights a critical phenomenon in LLM code generation capabilities. We select the highest score from five submissions as the model's total score, indicating that these models possess the knowledge required to solve problems correctly, yet struggle to produce correct solutions in a single attempt, resulting in low pass@1 rates. This finding underscores the importance of developing more reliable methods to guide LLMs' problem-solving capabilities as a crucial direction for future research.

4.5 Efforts to Prevent Data Leakage

The critical issue of LLMs overfitting (Jain et al., 2024)to benchmark leaderboards remains inadequately addressed. To ensure fair evaluation, we established a private leaderboard using 2024 ICPC World Finals problems with unreleased test cases, for which we constructed our own test cases, creating an unbiased assessment environment.

Our evaluation employs a two-tier system: models submitted to the public leaderboard undergo automatic testing on our private dataset. This approach reveals potential overfitting by comparing cross-leaderboard performance. Models with consistent results likely demonstrate authentic code generation capabilities, while performance disparities may indicate benchmark-specific overfitting.

5 Discussion

The Future Direction of Code LLM Capabilities The code LLM landscape is rapidly evolving with divergent approaches to balancing general coding capabilities and competitive programming skills. Claude 3.7 appears to prioritize general code generation over specialized algorithmic performance, possibly reflecting Anthropic's assessment of market demand for practical development assistance. Conversely, Gemini 2.5 Pro excels in both domains, suggesting that with sufficient scale and sophistication, the trade-off between these capabilities may be less constraining than previously thought. Therefore, general code abilities and competitive programming abilities in future LLMs are **not mutually exclusive directions**. LLM performance in programming competitions remains an important metric for gauging current progress toward AGI.

The Future Direction of Test-Time Scaling in Code LLM With the powerful performance demonstrated by Gemini 2.5-Pro and o4-mini (high), concerns have emerged about whether Test-time Scaling Laws, similar to Pre-training Scaling Laws (Hoffmann et al., 2022), are approaching their limits. Our experiments demonstrate that test-time scaling substantially impacts model performance. While these results are impressive, we have likely not reached the ceiling of potential improvements through such techniques. As models continue to scale in test time, we anticipate further advancements in reasoning efficiency (Dumitru et al.,

2025) and efficacy, and generating sophisticated algorithmic solutions.

6 Conclusion

In this work, we introduced HLCE, a challenging benchmark of 235 competitive programming problems from IOI and ICPC World Finals. Top models achieve only 15.1% and 11.4% pass@1 rates. Our benchmark includes standard and interactive programming challenges alongside a novel self-assessment task. By incorporating human competition data, we established metrics comparing LLMs with top-tier programmers, revealing substantial room for improvement. Test-time scaling law validation confirms current models have not reached performance ceilings, suggesting promising directions for advancing LLMs' reasoning in complex programming tasks. HLCE aims to drive progress toward code LLMs that reach the proficiency level of elite human competitors.

7 Limitations

Due to API pricing constraints and inference latency limitations, we could only generate 5 responses for each reasoning model. With more responses, LLMs would likely achieve better results in historical competitions. However, such a comparison might not be entirely fair, as within the specified competition time frame, human coding speed cannot match that of LLMs.

IOI problems require online submissions, resulting in longer submission processing times—approximately three minutes per problem on average. While this delay is generally acceptable for our evaluation purposes, it does add operational overhead to the evaluation process.

8 Ethical Considerations

We ensure that the distribution of each dataset complies with the corresponding licenses, all of which are listed below:

- IOI: Provided under "CC-BY-SA 4.0" license.
- ICPC World Finals: Provided under "CC-BY-SA 4.0" license.

For the new artifacts contributed in HLCE, including but not limited to the questions, test cases, and evaluation scripts, we make them available solely for research purposes. Users are permitted to use, modify, and share these annotations for academic and non-commercial research activities. Any other use, including commercial exploitation, is not permitted without explicit written permission from the authors.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Aider. 2024. Leaderboards - aider documentation.

Anthropic. 2025. Introducing the claude 3 model family.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Kuicai Dong, Aixin Sun, Jung-jae Kim, and Xiaoli Li. 2023. Open information extraction via chunks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15390–15404, Singapore. Association for Computational Linguistics.

Razvan-Gabriel Dumitru, Minglai Yang, Vikas Yadav, and Mihai Surdeanu. 2025. Copyspec: Accelerating llms with speculative copy-and-paste without compromising quality. *Preprint*, arXiv:2502.08923.

Thang Duong, Minglai Yang, and Chicheng Zhang. 2025. Improving the data-efficiency of reinforcement learning by warm-starting with llm. *Preprint*, arXiv:2505.10861.

Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev, Daniel Selsam, David Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, and 1 others. 2025. Competitive programming with large reasoning models. *arXiv preprint arXiv:2502.06807*.

Hugging Face. 2025. Open r1: A fully open reproduction of deepseek-r1.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and 1 others. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, and 1 others. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

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

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

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

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, and 1 others. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Xiangyang Li, Kuicai Dong, Yi Quan Lee, Wei Xia, Hao Zhang, Xinyi Dai, Yasheng Wang, and Ruiming Tang. 2025. CoIR: A comprehensive benchmark for code information retrieval models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22074–22091, Vienna, Austria. Association for Computational Linguistics.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Microsoft. 2024. Python tool for converting files and office documents to markdown.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

OpenAI. 2025. Introducing o3 and o4-mini.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Minglai Yang, Ethan Huang, Liang Zhang, Mihai Surdeanu, William Wang, and Liangming Pan. 2025b. How is llm reasoning distracted by irrelevant context? an analysis using a controlled benchmark. *Preprint*, arXiv:2505.18761.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, and 1 others. 2024. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*.

A Appendix

A.1 Dataset Statistic

The total quantity of filtered HLCE is presented in Table 6.

A.2 Problem Example

In Figures 4 and 5, we present sample problems from IOI and ICPC World Finals respectively.

A.3 API Cost

We calculated the actual API costs incurred during our evaluation process, with results presented in Table 7. Interestingly, the o4-mini model demonstrated remarkably low cost, emerging as the most

Year	ICPC World Finals	IOI
2010	-	8
2011	11	6
2012	12	6
2013	11	6
2014	12	6
2015	13	6
2016	13	6
2017	12	5
2018	5	6
2019	11	5
2020	14	6
2021	12	6
2022	10	6
2023	10	6
2024	-	6

Table 6: Problem Counts in ICPC and IOI by Year

Model	Cost (USD)
Gemini-2.5-Pro	758.11
DeepSeek-R1	359.55
o3-mini	239.70
claude-3.7-sonnet-thinking	118.44
o1-mini	109.86
o4-mini	77.90
claude-3-7-sonnet	39.48
gpt-4o-2024-05-13	25.67
claude-3-5-sonnet	23.37
chatgpt-4o-latest	19.98
deepseek-v3	16.98
gpt-4o-mini	1.02
Total	**1,790.06**

Table 7: API costs for our experiments.

economical among all inference models evaluated. This finding is particularly significant as it suggests a promising development trend for code-oriented LLMs: substantial performance improvements may be achievable with minimal financial investment.

The cost-effectiveness of the o4-mini model indicates that efficient architectures and training methodologies can significantly reduce computational expenses without compromising performance quality. This balance between cost and capability represents an important direction for the future development of code generation and inference systems, potentially democratizing access to powerful code LLMs for a wider range of applications and users.

A.4 Evaluation Details

A.4.1 Evaluation Parameters

For ICPC World Finals problems, following the LiveCodeBench, we set the execution timeout to 30 seconds without imposing memory limitations. For IOI problems, we utilize the web response information returned by the bot.

A.4.2 Used Prompt

In this section, we present all the task prompts utilized for the ICPC and IOI competitions, as illustrated in Figures 6, 7, and 8.

A.4.3 Evation Metric

Pass@K To evaluate the performance of code generation models, we employ the pass@k metric. This metric assesses a model's ability to generate at least one correct solution within k attempts.

Given a coding problem, we sample k candidate solutions from the model and evaluate each solution against test cases. The pass@k metric is defined as:

$$\text{pass@k} = \mathbb{E}_{x \sim D}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \quad (1)$$

where:

- x represents a coding problem sampled from dataset D
- n is the total number of solutions generated for each problem
- c is the number of correct solutions among the n generations
- $\binom{n-c}{k}$ and $\binom{n}{k}$ represent binomial coefficients

Intuitively, this formula calculates the probability of finding at least one correct solution when randomly selecting k samples from n generated solutions. When $n = k$, the formula simplifies to c/n, which is the fraction of correct solutions. For our experiments, we report pass@1, pass@5 to provide a comprehensive view of model performance across different sampling scenarios.

AUC (Area Under the Curve) The Area Under the Curve (AUC) measures the performance of binary classification models by quantifying the area under the Receiver Operating Characteristic (ROC) curve. Mathematically, AUC is defined as:

$$AUC = \int_0^1 TPR(FPR^{-1}(t))dt \quad (2)$$

where TPR is the True Positive Rate and FPR is the False Positive Rate.

In practice, AUC is computed using the trapezoidal rule:

$$AUC \approx \frac{1}{2} \sum_{i=1}^{n-1} \Big((FPR_{i+1} - FPR_i) \cdot (TPR_i + TPR_{i+1}) \Big) \quad (3)$$

The AUC represents the probability that a randomly selected positive instance will rank higher than a randomly selected negative instance. An AUC of 1.0 indicates perfect classification, while 0.5 suggests performance equivalent to random chance.

In the self-recognition task, we employ the AUC metric to evaluate whether LLMs can accurately identify if their generated answers are correct or incorrect. This metric provides a comprehensive measure of the model's ability to discriminate between its own correct and incorrect responses, effectively quantifying the model's self-recognition capabilities.

Example of ICPC World Finals problems

QuestionId:2021-H
QuestionName:Mining Your Own Business
Problem Description
John Digger is the owner of a large illudium phosdex mine. The mine is made up of a series of tunnels that meet at various large junctions. Unlike some owners, Digger actually cares about the welfare of his workers and has a concern about the layout of the mine. Specifically, he worries that there may be a junction which, in case of collapse, will cut off workers in one section of the mine from other workers (illudium phosdex, as you know, is highly unstable). To counter this, he wants to install special escape shafts from the junctions to the surface. He could install one escape shaft at each junction, but Digger doesn't care about his workers that much. Instead, he wants to install the minimum number of escape shafts so that if any of the junctions collapses, all the workers who survive the junction collapse will have a path to the surface.
Write a program to calculate the minimum number of escape shafts and the total number of ways in which this minimum number of escape shafts can be installed.
Input
The input consists of several test cases. The first line of each case contains a positive integer N ($N \leq 5 \cdot 10^4$) indicating the number of mine tunnels. Following this are N lines each containing two distinct integers s and t, where s and t are junction numbers. Junctions are numbered consecutively starting at 1. Each pair of junctions is joined by at most a single tunnel. Each set of mine tunnels forms one connected unit (that is, you can get from any one junction to any other).
The last test case is followed by a line containing a single zero.
Output
For each test case, display its case number followed by the minimum number of escape shafts needed for the system of mine tunnels and the total number of ways these escape shafts can be installed. You may assume that the result fits in a signed 64-bit integer.
Follow the format of the sample output.
Sample Input

```
9
1 3
4 1
3 5
1 2
2 6
1 5
6 3
1 6
3 2
6
1 2
1 3
2 4
2 5
3 6
3 7
0
```

Sample Output

```
Case 1: 2 4
Case 2: 4 1
```

Figure 4: Example of ICPC World Finals problem.

Example of IOI Problem

ProblemID:2024 D. Hieroglyphs
Time limit: 2 seconds
Memory limit: 1024 megabytes
Input/Output: standard
Problem Statement:
A team of researchers is studying the similarities between sequences of hieroglyphs. They represent each hieroglyph with a non-negative integer. To perform their study, they use the following concepts about sequences.
For a fixed sequence A, a sequence S is called a *subsequence* of A if and only if S can be obtained by removing some elements (possibly none) from A.
The table below shows some examples of subsequences of a sequence $A = [3, 2, 1, 2]$.

Subsequence	How it can be obtained from A
$[3, 2, 1, 2]$	No elements are removed.
$[2, 1, 2]$	[3, 2, 1, 2]
$[3, 2, 2]$	[3, 2, 1, 2]
$[3, 2]$	[3, 2, 1, 2] or [3, 2, 1, 2]
$[3]$	[3, 2, 1, 2]
$[]$	[3, 2, 1, 2]

...
Implementation details:
You should implement the following procedure:

```
std::vector<int> ucs(std::vector<int> A, std::vector<int> B)
```

- A: array of length N describing the first sequence.
- B: array of length M describing the second sequence.
- If there exists a universal common subsequence of A and B, the procedure should return an array containing this sequence. Otherwise, the procedure should return $[-1]$ (an array of length 1, whose only element is -1).

This procedure is called exactly once for each test case.
Input:
The sample grader reads in the following format:

- line 1: N M ($1 \leq N \leq 100\,000$, $1 \leq M \leq 100\,000$)
- line 2: $A[0]$ $A[1] \ldots A[N-1]$ ($0 \leq A[i] \leq 200\,000$)
- line 3: $B[0]$ $B[1] \ldots B[M-1]$ ($0 \leq B[j] \leq 200\,000$)

Output:
The sample grader prints in the following format:

- line 1: T
- line 2: $R[0]$ $R[1] \ldots R[T-1]$

Here, R is the array returned by `ucs` and T is its length.
Sample Input:

```
6 5
0 0 1 0 1 2
2 0 1 0 2
```

Sample Output:

```
4
0 1 0 2
```

Figure 5: Example of IOI problem.

Prompt for IOI code generation

````
You are now an expert contestant in the International Olympiad in Informatics (IOI). For most problems, please implement a C++ solution for the given problem with the following guidelines:
- You will be given a problem statement, test case constraints and example test inputs and outputs. Please reason step by step about the solution, then provide a complete implementation in C++.
- You should correctly implement the routine(s) described in Implementation Details, without reading or writing anything directly from stdin or to stdout, as input and output are passed through the implemented routines.
- Assume your code will be run on the OFFICIAL grader, and do not add a main, a sample grader, or any other test function unless it has been explicitly requested.
- IMPORTANT: When implementing functions required by the problem description that use notation like int[] or int64[] for function parameters, implement them as C++ std::vector types (e.g., vector<int> or vector<long long>), not as raw arrays or pointers.
- When declaring or implementing functions that are provided by the grader, use the EXACT same parameter types as specified. Do not use const references (const vector<int>&) or non-const references (vector<int>&) when the grader expects vector<int>, even if it would be more efficient.
- For multi-dimensional arrays like int[][], implement them as nested vectors (e.g., vector<vector<int») without references.
Please place your code between the following delimiters:

```cpp
// Your code will be placed here
```
````

Figure 6: Prompt for IOI code generation tasks.

Prompt for ICPC World Finals code generation

````
You are an expert Python programmer.
- You will be given a problem statement, test case constraints and example test inputs and outputs.
- You will generate a correct Python program that matches the specification and passes all tests
- Read the inputs from stdin solve the problem and write the answer to stdout (do not directly test on the sample inputs). Enclose your code within delimiters as follows. Ensure that when the python program runs, it reads the inputs, runs the algorithm and writes output to STDOUT.
Please place your code between the following delimiters:

```python
# YOUR CODE HERE
```
````

Figure 7: Prompt for ICPC World Finals code generation tasks.

Prompt for Self-recognition Tasks

````
You are now a code review expert. I will provide you with a programming problem description
along with a corresponding code implementation. Your task is to:
1. Carefully read and understand the problem requirements;
2. Analyze whether the code logic correctly fulfills the problem's requirements;
3. Determine whether the code can pass all test cases, including regular cases, edge cases,
and potential hidden tests;
4. In your response, first provide a detailed explanation of your analysis, including any
strengths, potential issues, or bugs you identify;
5. Finally, give your conclusion — it must be either of the following two options, and should
be wrapped using the delimiter below:

```answer
Yes
```

or

```answer
No
```
````

Figure 8: Prompt for Self-recognition tasks.