

# 000 001 002 003 004 005 006 007 008 009 010 AETHERCODE: EVALUATING LLMs’ ABILITY TO WIN IN PREMIER PROGRAMMING COMPETITIONS

005  
006  
007  
008  
009  
010  
011  
012  
013  
014  
015  
016  
017  
018  
019  
020  
021  
022  
023  
024  
025  
026  
027  
028  
029  
030  
031  
032  
033  
034  
035  
036  
037  
038  
039  
040  
041  
042  
043  
044  
045  
046  
047  
048  
049  
050  
051  
052  
053  
Anonymous authors  
Paper under double-blind review

## ABSTRACT

Competitive programming has emerged as a critical benchmark for evaluating the reasoning and coding capabilities of Large Language Models (LLMs). Despite impressive progress on existing benchmarks, we argue that current evaluations overstate model proficiency, masking a substantial gap between LLMs and elite human programmers. This gap arises from two key limitations: insufficient difficulty and scope of benchmark problems, and evaluation bias from low-quality test cases. To address these shortcomings, we present **AetherCode**, a new benchmark that draws problems from premier programming competitions such as IOI and ICPC, offering broader coverage and higher difficulty. AetherCode further incorporates comprehensive, expert-validated test suites built through a hybrid of automated generation and human curation, ensuring rigorous and reliable assessment. By combining challenging problem design with robust evaluation, AetherCode provides a more faithful measure of LLM capabilities and sets a new standard for future research in code reasoning.

## 1 INTRODUCTION

Competitive programming is widely regarded as a crucial benchmark for evaluating the reasoning and coding capabilities of Large Language Models (LLMs) (OpenAI et al., 2025). Solving complex competitive programming problems demands not only sophisticated reasoning abilities but also knowledge from diverse domains, including mathematics, data structures, and algorithms. Recent years have witnessed rapid advancements in the reasoning capabilities of LLMs, a key indicator of which is their success on a majority of existing code reasoning benchmarks. State-of-the-art models now achieve over 90% *Pass@1* accuracy on MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021), and over 80% on LiveCodeBench (Jain et al., 2025). These encouraging developments might lead one to ask: has competitive programming been mastered by LLMs?

In this paper, we argue that a significant gap still exists between the performance of LLMs and top-tier human competitors in programming contests. We propose that the perception of LLM dominance stems primarily from the limitations in the breadth and rigor of current code reasoning benchmarks, which are no longer sufficient to fully assess the capabilities of today’s increasingly powerful models. Specifically, we identify two main shortcomings in existing benchmarks:

- **Insufficient Difficulty and Scope.** Early benchmarks such as HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) consist of basic coding tasks, for instance, sorting or reversing a list, which present minimal reasoning challenges for state-of-the-art LLMs. More recent “competition-level” benchmarks often source problems from a limited set of websites. For example, LiveCodeBench (Jain et al., 2025) collects problems mainly from LeetCode and AtCoder, while CodeELO (Quan et al., 2025) and LiveCodeBench Pro (Zheng et al., 2025) originate solely from CodeForces. The problems from these websites have inherent limitations. LeetCode problems are generally easier and often require only the implementation of a single function rather than a complete program. CodeForces contests, which typically feature 5-7 problems within a 2-3 hour timeframe, constrain the design space for problem setters, for example, leading to a scarcity of problems that require complex, large-scale implementations.
- **Evaluation Bias from Low-Quality Test Cases.** Inaccurate verifiers introduce bias into the evaluation (Vendrow et al., 2025). The correctness of a piece of code is verified us-

Table 1: Comparison between AetherCode and other code reasoning benchmarks

Dataset	Difficulty	# Problems	Updates	Test Cases Construction	Source
HumanEval (Chen et al., 2021)	★	164	✗	Handcrafted	Original
MBPP (Austin et al., 2021)	★	974	✗	Handcrafted	Original
APPS (Hendrycks et al., 2021)	★★★	5,000	✗	Crawled	CodeForces, AtCoder <i>etc.</i>
USACO (Shi et al., 2024)	★★★	307	✗	Publicly accessible	USACO
CodeContests (Li et al., 2022)	★★★	165	✗	Mutation	CodeForces, AtCoder <i>etc.</i>
LiveCodeBench (Jain et al., 2025)	★★	1055	✓	Semi-automatic	LeetCode, AtCoder
CodeELO (Quan et al., 2025)	★★★	387	✓	-	CodeForces
LiveCodeBench Pro (Zheng et al., 2025)	★★★	584	✓	-	CodeForces
AetherCode	★★★	456	✓	G-V Agent & Experts	Premier Contests

ing a comprehensive set of test cases (input-output pairs). An incomplete test suite may fail to detect incorrect submissions, particularly those with subtle flaws, such as the mis-handling of corner cases or solutions that exceed time limits under specific, extreme conditions. Consequently, designing high-quality test cases is a huge challenge that requires a deep understanding of potential failure points, a skill typically honed through extensive competitive programming experience. Most past benchmarks lack sufficiently rigorous test cases. HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021), for instance, rely on a small number of handwritten test cases. Others, including EvalPlus (Liu et al., 2023), CodeContests (Li et al., 2022), and LiveCodeBench (Jain et al., 2025), employ naive test case generation pipelines, such as random mutation, which fall far short of the quality of expert-designed test suites. Furthermore, recent research (Wang et al., 2025b) has revealed issues with test case correctness itself; for example, many test cases in the CodeContests dataset do not adhere to the problem’s constraints, causing even correct solutions to fail. It is worth noting that some recent benchmarks, such as CodeELO (Quan et al., 2025) and LiveCodeBench Pro (Zheng et al., 2025), have attempted to leverage the official CodeForces judging service to indirectly access its high-quality, expert-crafted test cases. However, this approach presents two significant issues. First, it raises compliance risks, as CodeForces explicitly prohibits the use of crawlers on its judging interface. Second, this method is constrained by submission frequency limits, which impedes agile and flexible experimentation. Therefore, we contend that an open-source benchmark with high-quality, self-contained test cases remains critically important for the LLM community.

To address these challenges, we introduce AetherCode, a new benchmark with the following key contributions:

**Problem Curation from Top-Tier Competitions.** AetherCode is the first benchmark to systematically collect latest problems from premier programming competitions worldwide, including the Olympiad in Informatics (OI) and the International Collegiate Programming Contest (ICPC). Our process involved a comprehensive collection, meticulous cleaning, and format conversion of problems from PDF to a Markdown+LaTeX structure. Each problem statement was manually proofread for correctness, and a team of competitive programming experts annotated each problem with classification tags.

**High-Quality Test Case Generation.** We developed a hybrid methodology, combining automated generation with expert annotation, to create high-quality test cases for every problem. We evaluated the correctness and comprehensiveness of our test cases by validating them against a large corpus of collected solutions, enforcing a standard of zero false positives and zero false negatives.

This paper is organized as follows: Section 2 details the benchmark curation process. Section 3 presents our evaluation results. Section 4 presents some related work, and Section 5 concludes the paper with comments for future research.

## 2 BENCHMARK CURATION

This Section details the curation process of the AetherCode Benchmark. The overall curation process is illustrated in Fig. 1. Then, Sections 2.1 and 2.2 describe the specifics of problem collection

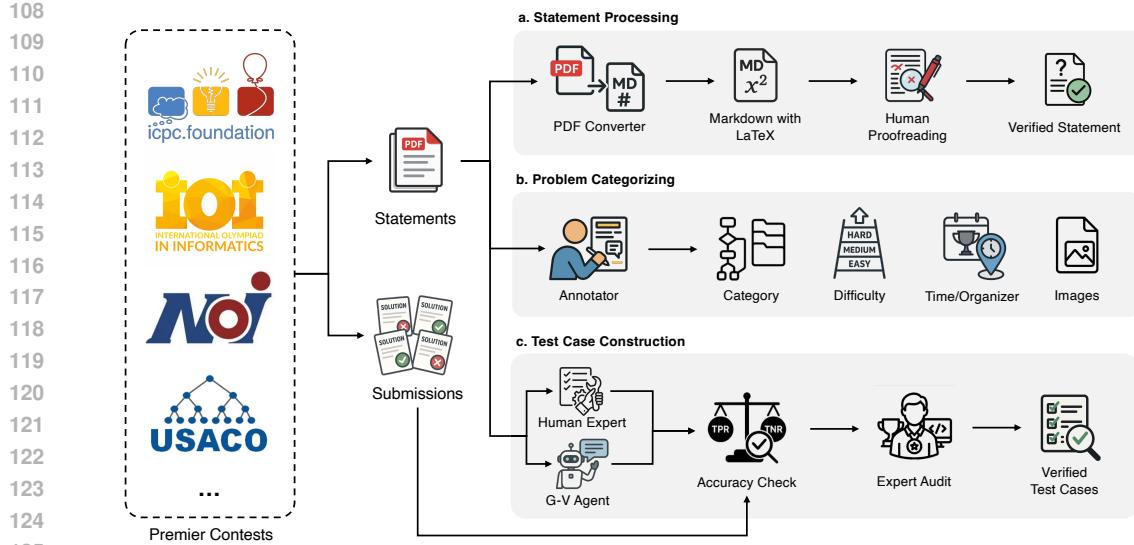


Figure 1: **Curation process of AetherCode.** (a) We begin by converting the collected problem statements from PDF to a Markdown+LaTeX format, which is then manually proofread for accuracy. (b) Each problem is then categorized by its algorithm type and difficulty level, and we also compile additional metadata, such as the time and the organizer of the contest. (c) To ensure quality, we use a G-V Agent in conjunction with human experts to annotate test cases. The accuracy of these test cases is then evaluated against a collected set of solutions and further audited by gold medalists and professional problem setters.

and categorizing, respectively. Section 2.3 explains how we construct high-quality test cases for each problem.

## 2.1 PROBLEM COLLECTION

We source our problems from premier programming competitions worldwide rather than from online programming websites. Based on their target audience, these competitions can be broadly categorized into two main series: the Olympiad in Informatics (OI) series, which is aimed at pre-college school students, and the International Collegiate Programming Contest (ICPC) series, which is designed for college students.

**OI Series.** The Olympiad in Informatics is a series of competitions aimed at popularizing computer science knowledge among [high-school](#) students and cultivating outstanding talents in computer science. The OI competitions usually require participants to solve algorithm-related problems by programming. Take the International Olympiad in Informatics (IOI), the top-level event of OI, as an example. Each contestant competes individually, and each country can send up to 4 players. During the two-day competition, players need to independently solve 3 problems within 5 hours each day, mainly using C++. Furthermore, various countries and regions host their own national or regional OI competitions, such as the National Olympiad in Informatics (NOI) in China and the USA Computing Olympiad (USACO) in the United States. Top-performing contestants in these competitions earn the opportunity to advance to the IOI.

**ICPC Series.** The ICPC is the oldest, largest, and most prestigious university-level programming contest in the world. Each team consists of up to 3 students and uses one computer to solve 10 - 13 problems in 5 hours, using programming languages such as C, C++, Java, or Python. The team that correctly solves the most problems with the least total time wins.

The world is divided into several regions for the ICPC. In Europe, there are Central Europe (CERC), North Europe (NWERC), South-East Europe (SEERC), and South-West Europe (SWERC) regions. Other regions include Asia-Pacific, Asia East Continent, North America, Latin America, Africa, and Arab region, etc. The ICPC is a multi-tiered event. First, there are **regional contests** held worldwide

162 from September to November each year. The top-performing teams in the regional contests advance  
 163 to the **regional finals or championships**. Then, the best teams from these finals or championships  
 164 qualify for the ICPC **World Finals**, which is usually held from April to June each year. This is  
 165 the highest-level stage of the ICPC, where the best teams from around the world compete for the  
 166 championship.

167 In addition to the official ICPC events, we also incorporated problems from other large-scale and  
 168 renowned collegiate programming contests, such as the China Collegiate Programming Contest  
 169 (CCPC).

170 For each problem, we collected the following components:

- 172 • **Problem Statement.** The statement typically comprises a title, a detailed problem description,  
 173 input/output specifications, sample inputs and outputs with explanations, data range  
 174 constraints, and time/memory limits. The majority of the problem statements was originally  
 175 in PDF format. To enhance comprehension for LLMs, we converted these PDFs into  
 176 a Markdown format with LaTeX for mathematical notations. Each converted file was then  
 177 manually proofread to ensure its accuracy.
- 178 • **Solutions.** We curated a collection of over 30,000 human-written solutions for these problems,  
 179 encompassing both correct and incorrect submissions. For each problem, we ensured  
 180 a minimum of 5 correct and 20 incorrect solutions. The primary purpose of collecting  
 181 these solutions is to evaluate the quality of the subsequently generated test cases, a process  
 182 detailed in Section 2.3.
- 183 • **Test Cases.** A minority of the competitions, e.g., USACO, publicly released their official  
 184 test cases, which we collected and standardized. For problems where official test cases  
 185 were not available, we constructed our high-quality test cases. The methodology for this  
 186 construction is described in Section 2.3.
- 187 • **Metadata.** We also gathered auxiliary information, such as the date of the competition (for  
 188 decontamination purposes) and human contestant performance data (to facilitate difficulty  
 189 assessment), among other available data points.

190 The data characteristics and statistics of AetherCode v1 are presented in Table 2.

## 192 2.2 PROBLEM CATEGORIZATION

194 Beyond curating problems, an equally critical step in constructing AetherCode was the systematic  
 195 categorization of each problem to ensure comprehensive coverage and facilitate fine-grained evalua-  
 196 tion. To this end, we adopted a multi-dimensional categorization framework designed with the input  
 197 of competitive programming experts:

199 **Difficulty Segmentation.** Problems were divided into four levels of difficulty: *Easy*, *Medium*, *Hard*,  
 200 and *Extreme*. This classification was guided by expert judgment as well as official contest results.  
 201 Most of the problems are distributed roughly evenly among *Easy*, *Medium*, and *Hard*. Notably,  
 202 problems that no human contestant was able to solve during a competition were specially classified  
 203 as *Extreme*, representing challenges that push the boundaries of algorithmic reasoning. The number  
 204 of problems under each difficulty level is presented in Fig. 2. This difficulty classification is judged  
 205 entirely from the perspective of humans rather than being classified by LLM’s performance. This  
 206 is because we want to provide a perspective to study how the difficulty for LLMs differs from the  
 207 difficulty in the eyes of humans. Specifically, we rank problems within the same contest based  
 208 on the number of participants who successfully solved them. For contests without leaderboards,  
 209 as well as for determining the relative difficulty order across different contests, we rely on expert  
 210 evaluation. Finally, based on the overall difficulty ranking of all problems, we divide the dataset into  
 211 three roughly equal categories: *Easy*, *Medium*, and *Hard*.

212 **Temporal and Contextual Dimensions.** Each problem was annotated with metadata to enable both  
 213 decontamination and longitudinal analysis of model performance: (1) **Date of the contest**, allowing  
 214 chronological tracking of trends in problem design and model capabilities. (2) **Organizer and**  
 215 **competition type**, primarily distinguishing between Olympiad in Informatics (OI) and International  
 Collegiate Programming Contest (ICPC) series. (3) **Competition scope**, categorizing contests as  
 regional-level, national-level, or worldwide.

216 **Problem Properties.** Some problems require additional considerations beyond a standard input–output interface: (1) Problems dependent on visual or image-based input were excluded from  
 217 the benchmark. (2) Problems requiring special judges (custom checkers) were explicitly labeled to  
 218 ensure proper handling during evaluation.  
 219

220 **Algorithmic and Domain Categories.** To capture the breadth of algorithmic knowledge tested  
 221 in programming contests, we implemented a hierarchical taxonomy as shown in Appendix B. The  
 222 first level consists of ten categories: Algorithm Basics, Search, Dynamic Programming, Strings,  
 223 Mathematics, Data Structures, Graph Theory, Computational Geometry, Common Techniques, and  
 224 Problems on Trees. The number of problems corresponding to these ten major categories is shown  
 225 in the Figure 2. The second level has 144 categories, which are presented in Appendix B. It includes  
 226 more detailed algorithm tags. For example, the major category “Mathematics” contains several sub-  
 227 categories such as Number Theory, Linear Algebra, Probability, Game Theory, Combinatorics, and  
 228 Polynomials. Problems can belong to multiple categories to reflect their cross-disciplinary nature.  
 229

230 This structured categorization enables targeted evaluation of model strengths and weaknesses while  
 231 also ensuring that AetherCode serves as a scalable resource for future research. In particular, it  
 232 allows progress to be tracked across difficulty levels, problem types, and algorithmic domains, pro-  
 233 viding a more comprehensive understanding of model capabilities.  
 234

### 235 2.3 TEST CASE CONSTRUCTION

236 Recent studies (Liu et al., 2023; Wang et al., 2025b) have highlighted concerns regarding the quality  
 237 of test cases in several existing code datasets. For instance, benchmarks such as MBPP (Austin  
 238 et al., 2021) and HumanEval (Chen et al., 2021) include only a limited number of handwritten test  
 239 cases per problem. Others, like CodeContests (Li et al., 2022) and EvalPlus (Liu et al., 2023),  
 240 rely on naive methods such as mutation to generate test cases. Consequently, such test cases are  
 241 insufficient for comprehensively evaluating the correctness and efficiency of a program. Therefore,  
 242 we contend that the quality of test case construction is a critical factor determining the overall quality  
 243 of a benchmark.  
 244

245 Notably, some recent benchmarks (Quan et al., 2025; Zheng et al., 2025) directly utilize the Code-  
 246 Forces’s judging service for evaluation. This approach allows them to indirectly access high-quality  
 247 test cases created by professional problem setters, thereby circumventing the challenge of test case  
 248 construction. However, this method presents potential compliance risks, as CodeForces explicitly  
 249 prohibits the use of crawlers on its judging interface. Furthermore, this approach is constrained by  
 250 submission frequency limits, which impedes agile and flexible evaluation. Therefore, we argue that  
 251 a benchmark equipped with its own high-quality test cases remains critically important for the LLM  
 252 community.  
 253

254 To ensure AetherCode possesses sufficiently high-quality test cases, we approached the task from  
 255 two perspectives. First, we established more stringent evaluation criteria for test case quality, which  
 256 is presented in Section 2.3.1. Second, we employed a hybrid approach, combining automated gen-  
 257 eration with expert annotation, to construct the test cases, which are presented in Sections 2.3.2 and  
 258 2.3.3. A detailed procedure of test case generation is presented in Appendix C.  
 259

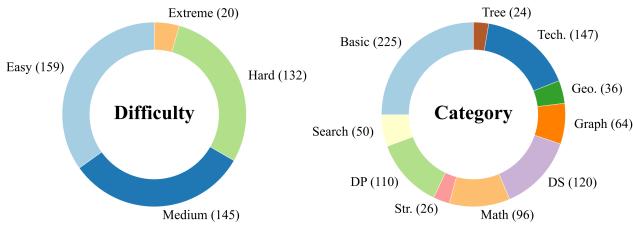
#### 260 2.3.1 TEST CASE QUALITY ASSESSMENT

261 Previous research on test case quality has predominantly focused on quantity, operating under the  
 262 assumption that a greater number of test cases correlates with higher quality (Li et al., 2022; 2023).  
 263 However, recent studies (Wang et al., 2025b) indicate that quantity is not a direct proxy for quality.  
 264 This discrepancy arises from two primary issues. First, test cases in some older datasets, despite their  
 265 volume, suffer from significant correctness issues, often violating the problem’s explicit constraints.  
 266 Second, conventional test case generation methods that merely amass large volumes of random data  
 267 fail to provide adequate coverage of various special and corner cases.  
 268

269 Consequently, we depart from evaluating test cases by their quantity and instead propose a direct  
 270 assessment of their ability to discriminate between correct and incorrect solutions. In our framework,  
 271 we conceptualize the entire test suite for a problem as a binary classifier, that is, a classifier that  
 272 distinguishes between correct and incorrect solutions. We then evaluate the performance of this  
 273

270  
271 Table 2: Data characteristics of  
272 AetherCode v1 (2401-2505)

273 CATEGORY	274 METRIC	275 COUNT
276 <b>Year</b>	# 2024	400
	# 2025	56
277 <b>Origin</b>	# OI	76
	# ICPC	380
<b>278 Test Cases</b>	Avg. Tests	47.15
<b>280 Categories</b>	# Categories	10
	# Tags	144

282  
283 Figure 2: Difficulty and category distributions of the Aether-  
284 Code v1 (2401-2505). The definitions of the category abbreviations are in Table 4.285  
286 classifier using a large, curated collection of both correct and incorrect submissions. We adopt the  
287 True Positive Rate (TPR) and True Negative Rate (TNR) as our primary evaluation metrics.

288  
289  
290

$$TPR = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} = \frac{\text{Number of Passed Correct Solutions}}{\text{Number of Correct Solutions}} \quad (1)$$

$$TNR = \frac{\text{True Negative}}{\text{True Negative} + \text{False Positive}} = \frac{\text{Number of Rejected Incorrect Solutions}}{\text{Number of Incorrect Solutions}} \quad (2)$$

291  
292 The TPR measures the **correctness** of the test cases; a high TPR indicates that correct solutions are  
293 not erroneously failed, which is expected when the test cases themselves are valid. Conversely, the  
294 TNR measures the **comprehensiveness** or **coverage** of the test cases, quantifying their ability to  
detect (or “hack”) incorrect solutions.295  
296 By employing a hybrid approach that combines automated generation with expert curation, we have  
297 achieved a 100% TPR and 100% TNR on our collected solution set. This signifies that all collected  
298 correct solutions pass our test cases, while all collected incorrect solutions are successfully rejected.  
299 To the best of our knowledge, AetherCode is the first benchmark that sets such a high standard for  
300 test cases.301  
302 

### 2.3.2 AUTOMATIC CONSTRUCTION OF TEST CASES

303  
304 We employed the Generator-Validator (G-V) Agent System (Wang et al., 2025b) to automatically  
305 construct test cases for these problems. This is a multi-agent system composed of two interacting  
306 agents: a validator and a generator. The generator agent writes a test case generator program to  
307 produce diverse test cases, including random and various corner cases. The validator agent writes a  
308 validator program to ensure that the test cases produced by the generator are correct and adhere to  
the problem’s constraints.309  
310 Previous research (Wang et al., 2025b) has pointed out that some past code datasets blindly increased  
311 the quantity of test cases while ignoring their validity, thereby introducing significant bias into eval-  
312 uations. Therefore, the validator plays a crucial role in ensuring the correctness of the test cases. To  
313 further guarantee quality, we have added a manual human-in-the-loop step to review and correct the  
314 validator programs, ensuring that all of our test cases are valid.315  
316 In this test case generation task, the G-V agent system alone achieves a TNR of 89.9%. Furthermore,  
317 due to the incorporation of additional human verification for the validator, it attains a TPR of 100%.  
318 Recognizing that this Automatic Construction phase could not achieve a 100% TNR on its own, we  
319 introduced an additional expert annotation stage to further strengthen the test cases.320  
321 

### 2.3.3 EXPERT ANNOTATION OF TEST CASES

322  
323 To this end, we recruited 67 competitive programming experts. The majority of them hold Code-  
324 forces ratings above 2000, with a few experts exceeding 2600 and achieving the title of International  
325 Grandmaster. These experts were tasked with constructing targeted test cases specifically designed  
326 to fail the various incorrect solutions we had collected. These manually crafted test cases were then  
327 merged with the automatically generated ones to form the final test suite.

324  
 325 Table 3: Performance comparison between reasoning models and non-reasoning models on Aether-  
 326 Code v1 (%), 2401-2505). The **Difficulty** and **Year** columns show the models’ pass@1 scores on  
 327 problems of varying difficulty levels and from different years. The **Pass@N** column displays the  
 328 models’ Pass@1, Pass@2, and Pass@4 scores.

Model	DIFFICULTY				YEAR		PASS@N		
	Easy	Medium	Hard	Extreme	2024	2025	1	2	4
<i>Reasoning Models</i>									
o4-mini-high	<b>65.3</b>	<b>32.1</b>	8.0	<b>3.8</b>	<b>35.8</b>	<b>32.6</b>	<b>35.5</b>	<b>43.0</b>	<b>46.6</b>
Gemini-2.5-Pro	60.1	28.6	<b>8.5</b>	2.5	33.7	25.0	32.7	39.8	46.0
Seed-1.6-Thinking-0715	53.9	20.2	4.7	0	28.3	14.7	26.6	33.0	38.5
DeepSeek-R1-0528	46.2	16.0	3.8	0	23.4	14.3	22.3	27.4	32.4
Qwen3-235B-A22B-Thinking-2507	43.1	18.6	4.0	1.3	23.6	11.6	22.2	28.9	36.0
Gemini-2.5-Flash	42.1	15.2	2.7	0	22.0	8.0	20.3	24.5	28.5
GLM-4.5	40.1	14.3	2.7	0	20.6	9.8	19.3	24.9	29.2
Qwen3-235B-A22B	37.6	12.4	1.9	0	19.1	7.1	17.6	21.7	25.2
Qwen3-32B	34.8	10.9	2.7	0	17.7	6.7	16.3	20.4	23.9
<b>Claude-4.5-Sonnet-thinking</b>	<b>36.8</b>	<b>8.8</b>	<b>2.2</b>	<b>0</b>	<b>17.1</b>	<b>10.3</b>	<b>16.3</b>	<b>19.8</b>	<b>23.3</b>
Claude-4-Opus-thinking	30.0	5.2	1.0	0	13.1	7.6	12.4	15.6	18.2
Qwen3-8B	23.7	4.8	0.8	0	11.1	2.7	10.0	13.0	15.5
<i>Non-Reasoning Models</i>									
GPT-4.1	23.9	5.7	1.1	0	11.3	4.5	<b>10.5</b>	13.2	15.3
Kimi-K2	23.1	4.7	1.0	0	10.6	4.0	9.8	12.2	14.5
DeepSeek-V3-0324	20.8	4.0	0	0	8.9	5.4	8.5	10.5	12.3
Qwen3-Coder-480B-A35B	19.7	2.2	0.6	0	8.6	1.8	7.7	9.9	11.8
Claude-4-Sonnet-nothinking	18.4	2.6	0.8	0	7.9	4.5	7.5	9.1	11.0
GPT-4o	11.6	1.0	0.2	0	4.9	1.3	4.4	5.6	7.0

348 Furthermore, we recognized that for certain problems with a limited number of collected incorrect  
 349 solutions (fewer than 50), achieving a 100% TNR might not sufficiently guarantee the robustness  
 350 of the test cases. To address this, we subjected the test cases for all problems to a manual quality  
 351 audit by a specialized review team. Each member of this elite team holds at least three ICPC gold  
 352 medals and has a minimum of two years of experience in competitive programming problem-setting.  
 353 Their deep understanding of potential pitfalls and common errors in each problem allows them to  
 354 leverage their extensive experience to further ensure the quality and comprehensiveness of the test  
 355 cases. **Specifically, this elite team further supplements missing corner cases and additionally writes**  
 356 **various incorrect and inefficient solutions to verify the comprehensiveness of the test cases.**

357 Additionally, for problems that accept multiple valid outputs, customized judging scripts (a.k.a.  
 358 checker, or special judge) were provided and thoroughly reviewed by these experts to ensure correct  
 359 evaluation.

### 3 EVALUATION

364 Our evaluation includes 11 reasoning models and 6 non-reasoning models. The reasoning mod-  
 365 els comprise o4-mini-high (OpenAI, c), Gemini-2.5-Pro/Flash (Comanici et al., 2025),  
 366 Seed-1.6-thinking (Chen et al., 2025a), DeepSeek-R1 Guo et al. (2025), GLM-4.5 (Zeng  
 367 et al., 2025), Claude-4-Opus-thinking and Qwen3 (Yang et al., 2025a), among others. The non-  
 368 reasoning models consist of GPT-4.1 (OpenAI, a), GPT-4o (OpenAI, b), Kimi-K2 (Kimi-Team  
 369 et al., 2025), DeepSeek-V3 (Liu et al., 2024), Claude-4-Sonnet (without thinking), and  
 370 Qwen3-Coder. All models are configured with a maximum output length of 32,768 tokens. Each  
 371 model is evaluated four times in each problem, and the average numbers are reported. Detailed  
 372 settings of the experiment are presented in Appendix A.

#### 3.1 MAIN RESULT

374 Table 3 presents a comprehensive performance evaluation of several prominent models on Aether-  
 375 Code. For full results, please refer to the online leaderboard. The analysis yields the following key  
 376 conclusions:

378 Table 4: Performance comparison (Pass@1) between reasoning models and non-reasoning models  
 379 across 10 major categories: Algorithm Basics (Basic), Search, Dynamic Programming (DP), Strings  
 380 (Str.), Mathematics (Math), Data Structures (DS), Graph Theory (Graph), Computational Geometry  
 381 (Geo.), Common Techniques (Tech.), and Problems on Trees (Tree).

Model	Basic	Search	DP	Str.	Math	DS	Graph	Geo.	Tech.	Tree
<i>Reasoning Models</i>										
o4-mini-high	<b>38.1</b>	<b>28.5</b>	<b>27.7</b>	<b>35.6</b>	<b>31.8</b>	<b>25.8</b>	<b>28.5</b>	<b>27.1</b>	<b>26.9</b>	<b>7.3</b>
Gemini-2.5-Pro	36.1	24.5	24.6	29.8	31.5	25.4	26.2	18.1	23.0	<b>7.3</b>
Seed-1.6-Thinking	32.2	17.0	17.3	26.0	24.2	17.9	18.8	12.5	19.2	1.0
DeepSeek-R1-0528	26.3	16.0	14.6	23.1	19.3	16.3	15.6	10.4	13.8	7.3
Qwen3-235B-A22B-Thinking-2507	26.2	14.5	15.0	20.2	21.1	14.8	15.6	11.8	15.1	4.2
Gemini-2.5-Flash	24.1	16.5	11.8	19.2	16.7	16.3	17.2	13.2	11.4	4.2
GLM-4.5	22.8	14.0	13.0	21.2	15.6	12.9	13.7	10.4	15.1	2.1
Qwen3-235B-A22B	22.2	13.0	8.4	20.2	13.5	11.0	12.5	11.1	9.4	4.2
Qwen3-32B	19.7	11.5	10.9	18.3	14.1	11.0	9.4	6.9	11.2	0
<b>Claude-4.5-Sonnet-thinking</b>	<b>20.7</b>	<b>11.5</b>	<b>8.2</b>	<b>17.3</b>	<b>9.1</b>	<b>9.4</b>	<b>10.6</b>	<b>11.1</b>	<b>11.4</b>	<b>0</b>
Claude-4-Opus-thinking	16.0	10.0	5.7	17.3	6.3	8.3	7.0	8.3	7.7	0
Qwen3-8B	13.3	9.0	3.9	15.4	7.6	7.9	6.3	1.4	4.9	1.0
<i>Non-Reasoning Models</i>										
GPT-4.1	13.9	9.5	3.4	19.2	4.2	8.3	5.5	6.3	6.0	0
Kimi-K2	13.7	7.5	3.6	15.4	7.0	8.1	6.6	0.7	3.6	0
DeepSeek-V3-0324	12.1	7.0	1.8	14.4	3.9	6.3	4.3	0	3.6	0
Qwen3-Coder-480B-A35B	11.1	5.5	1.8	14.4	4.2	5.2	4.3	1.4	2.9	1.0
Claude-4-Sonnet-nothinking	10.9	8.0	1.8	13.5	2.6	5.0	3.5	2.1	3.4	0
GPT-4o	7.2	4.5	0.7	11.5	1.6	2.9	0.4	0	1.5	0

401 **Significant Performance Gap between Models.** o4-mini-high and Gemini-2.5-Pro de-  
 402 liver exceptional performance, establishing an elite tier with a significant gap over other models.  
 403 They are notably two of the three models capable of tackling the "Extremely Difficult" problems.  
 404 This consistent, substantial lead across all difficulty tiers underscores the high degree of discrimina-  
 405 tion provided by the AetherCode benchmark.

406 **Reasoning Models Comprehensively Outperform Non-Reasoning Models.** As anticipated, rea-  
 407 soning models demonstrate markedly superior performance compared to non-reasoning models. For  
 408 instance, models from the Qwen3 series, such as Qwen3-32B, outperform several non-reasoning  
 409 models despite having fewer parameters. More notably, even with four sampling attempts (*Pass@4*),  
 410 the performance of non-reasoning models still falls short of that achieved by reasoning models.  
 411 This phenomenon indicates that for complex tasks like coding competitions, the solution space ex-  
 412 ploration capabilities of non-reasoning models are constrained, making it difficult to find correct  
 413 solutions through limited sampling. This bottleneck is particularly pronounced in weaker models.

414 **Top-Tier Models Exhibit Great Exploration Potential.** A comparison of *Pass@1* and *Pass@4*  
 415 scores reveals that increasing the number of samples yields a more substantial performance improve-  
 416 ment for top-tier models. For example, o4-mini-high's score improved by 11.1% (from 35.5%  
 417 to 46.6%), whereas the weaker Qwen3-32B only saw a gain of 7.6% (from 16.3% to 23.9%).  
 418 Particularly noteworthy is Gemini-2.5-Pro, which achieved a remarkable performance increase  
 419 of 13.3% (from 32.5% to 46.0%). This demonstrates its vast exploration potential in solving com-  
 420 plex programming problems, enabling it to generate more diverse and high-quality solutions through  
 421 multiple attempts.

### 423 3.2 PERFORMANCE ACROSS ALGORITHMS

425 The performance comparison in Table 4 reveals a significant differentiation in model capabilities  
 426 across various problem categories. All models, regardless of being reasoning or non-reasoning  
 427 types, uniformly excel at pattern-based tasks such as "Basic Algorithms" and "Strings". However,  
 428 their limitations become equally apparent when handling highly abstract problems. Most mod-  
 429 els struggle to tackle "Computational Geometry" and "Tree Structures", with the performance of  
 430 o4-mini-high in computational geometry being a notable exception. Furthermore, the short-  
 431 comings of non-reasoning models are particularly pronounced, as their capability bottlenecks extend  
 into domains that also demand deep logic and abstract thinking, such as "Dynamic Programming"

432 and “Mathematics”. It is worth noting that, due to the inconsistent distribution of problems across  
 433 categories, individual categories (such as Tree) may happen to be particularly difficult, resulting in  
 434 lower model scores. The difficulty distribution for each category is presented in Appendix B.  
 435

436 Generally speaking, models with higher overall scores also tend to be stronger across nearly every  
 437 subcategory, with o4-mini-high ranking first in all of them. However, this analysis also allows  
 438 us to identify the weaknesses of certain models. For example, while GPT-4.1 has the highest overall  
 439 score among the non-reasoning models, its performance on mathematical problems is significantly  
 440 weaker.  
 441

### 442 3.3 DIAGNOSIS OF FAILURE REASONS

443 We categorize all model failure cases into four types. *Wrong Answer* means the program outputted  
 444 an incorrect result. *Time Limit Exceeded* means the program failed to output an answer within the  
 445 given time limit. *Runtime Error* means the program encountered an error during runtime, including  
 446 Segmentation Error, exceeding the memory limit, etc. *Compile Error* means the program has a syn-  
 447 tax error and could not be successfully compiled. The statistical results are presented in Appendix  
 448 E Table 8.  
 449

450 For the majority of models, the primary error type is *Wrong Answer*, accounting for approximately  
 451 70% to 80% of cases, followed by *Time Limit Exceeded*. The Claude series is slightly different,  
 452 with *Wrong Answer* and *Time Limit Exceeded* each accounting for roughly half of the errors. We  
 453 conducted a study on the failure cases of the Claude models and found that, on difficult problems,  
 454 they tend to design algorithms that are correct but inefficient, rather than prioritizing adherence to  
 455 the problem’s time complexity constraints.  
 456

457 *Compile Error* rates vary significantly across different models. Most models maintain a *Compile*  
 458 *Error* rate within 10%, with the Claude series achieving the lowest. However, some models ex-  
 459 hibit particularly high *Compile Error* rates, such as GLM-4.5. Our analysis of GLM-4.5’s *Compile*  
 460 *Error* cases revealed that over half were caused by the model using the incorrect programming lan-  
 461 guage; for example, it writes a Python program while being instructed to use C++. This indicates a  
 462 deficiency in GLM-4.5’s ability to follow programming language instructions.  
 463

464 To further analyze the causes of model failures, we conducted a more granular attribution of er-  
 465 ror types. We performed a qualitative analysis of reasoning cases from o4-mini-high. The  
 466 primary failure reasons identified include: incorrect algorithmic logic, failure to handle corner  
 467 cases, insufficient algorithmic efficiency, and implementation errors. Furthermore, we found that  
 468 o4-mini-high sometimes acknowledges its inability to solve a problem rather than providing an  
 469 incorrect answer. Relevant problems and corresponding cases are provided in the Appendix E.  
 470

## 471 4 RELATED WORK

### 472 4.1 CODE BENCHMARKS

473 Coding ability is one of the important capabilities of LLMs. How to evaluate the coding ability of  
 474 LLMs has also received widespread attention from researchers in recent years. Existing code bench-  
 475 marks can be roughly divided into three categories: basic benchmarks, code reasoning benchmarks,  
 476 and software-engineering (SWE) benchmarks.  
 477

478 Representative of the basic benchmarks are HumanEval (Chen et al., 2021), MBPP (Austin et al.,  
 479 2021), CoderEval (Yu et al., 2024), PPM (Chen et al., 2024), DynaCode (Hu et al., 2025), and  
 480 DyCodeEval (Chen et al., 2025b), which contain some fundamental programming tasks such as  
 481 sorting and simple sequence operations. Because they are relatively basic, they are also suitable as  
 482 observation metrics for model capabilities during the pre-training stage.  
 483

484 Code reasoning benchmarks are primarily composed of competitive-level programming tasks that  
 485 simultaneously evaluate a model’s reasoning and coding abilities, testing its capacity for deep rea-  
 486 soning. Representative examples include CodeContests (Li et al., 2022) and LiveCodeBench (Jain  
 487 et al., 2025). In this paper, we mainly focus on code reasoning tasks.  
 488

486 Software-engineering (SWE) benchmarks mainly focus on repository-level engineering code tasks,  
 487 examining the model’s agentic capabilities, environmental interaction, code comprehension, task  
 488 planning, instruction following, and long-context abilities. Representative examples in this category  
 489 include SWE-bench (Jimenez et al., 2024), SWE-bench Verified (Chowdhury et al., 2025), Multi-  
 490 SWE-bench (Zan et al., 2025), and EvoCodeBench (Li et al., 2024).

491

## 492 4.2 CODE REASONING BENCHMARKS

493

494 Code reasoning benchmarks primarily consist of competition-level programming problems and are  
 495 used to evaluate the deep reasoning capabilities of LLMs. Many existing benchmarks collect prob-  
 496 lems from online programming websites like LeetCode and CodeForces, including CodeContests  
 497 (Li et al., 2022) (from Aizu, AtCoder, Codechef, CodeForces, HackerEarth), LiveCodeBench (Jain  
 498 et al., 2025) (from AtCoder and LeetCode), CodeELO (Quan et al., 2025) (from CodeForces), Live-  
 499 CodeBench Pro (Zheng et al., 2025) (from CodeForces), and ProBench (Yang et al., 2025b) (from  
 500 CodeForces, Luogu, NowCoder). Some efforts have also been made to gather problems from ma-  
 501 jor competitions, including USACO Bench (Shi et al., 2024), LLM-Pros (Hossain et al., 2025),  
 502 OJBench (Wang et al., 2025a), and ICPC-Eval (Xu et al., 2025); however, these are limited to a few  
 503 specific contests, and some rely on outdated data, posing a significant risk of data contamination.  
 504 For example, ICPC-Eval only includes 11 ICPC contests from 2023 to 2024; USACO Benchmark  
 505 includes USACO problems from 2011 to 2023; OJBench only includes 4 ICPC contests with NOI  
 506 problems from 2016 to 2023; LLM-Pros includes 14 ICPC contests from 2011 to 2024. To our  
 507 knowledge, AetherCode is the first benchmark to comprehensively collect latest problems from pre-  
 508 mier competitions around the world, surpassing previous work in both breadth and depth.

509

## 510 5 CONCLUSION

511

512 In this paper, we introduced AetherCode, a challenging, rigorously evaluated benchmark purpose-  
 513 built to assess LLMs’ coding and reasoning capabilities. AetherCode distinguishes itself by sourc-  
 514 ing all its problems from premier global programming competitions, including OI series and ICPC  
 515 series, which ensures a high degree of challenge and relevance. Furthermore, it features a com-  
 516 prehensive and meticulously validated suite of test cases, created through a hybrid model of automated  
 517 generation and expert curation. By validating against a dataset of over 30,000 human submissions,  
 518 our test suite achieves 100% TPR and 100% TNR on our collected solution set, guaranteeing excep-  
 519 tional accuracy and reliability in evaluation.

520

521 Our comprehensive evaluation of several leading-edge models on AetherCode yielded critical in-  
 522 sights. We observed a significant performance disparity among models, with top performers like  
 523 o4-mini-high and Gemini-2.5-Pro establishing a distinct upper tier. Reasoning models  
 524 demonstrated a clear and consistent advantage over their non-reasoning counterparts across all diffi-  
 525 culty levels, highlighting the crucial role of logical deduction in solving complex algorithmic prob-  
 526 lems. Overall, even the most advanced models today can only solve a small fraction of problems  
 527 in AetherCode. This indicates that current LLMs still have considerable room for improvement in  
 528 reasoning and coding, and there remains a significant gap compared to top human experts.

529

## 530 REFERENCES

531

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.

532

Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiang-  
 533 peng Wei, Wenyuan Xu, Yufeng Yuan, Yu Yue, Lin Yan, Qiying Yu, Xiaochen Zuo, Chi Zhang,  
 534 Ruofei Zhu, Zhecheng An, Zhihao Bai, Yu Bao, Xingyan Bin, Jiangjie Chen, Feng Chen, Hong-  
 535 min Chen, Riwei Chen, Liangqiang Chen, Zixin Chen, Jinsong Chen, Siyan Chen, Kaiyuan Chen,  
 536 Zhi Chen, Jin Chen, Jiecao Chen, Jinxin Chi, Weinan Dai, Ning Dai, Jiahui Dai, Shihan Dou,  
 537 Yantao Du, Zhengyin Du, Jianhui Duan, Chen Dun, Ting-Han Fan, Jiazhan Feng, Junda Feng,  
 538 Ziyuan Feng, Yuwei Fu, Wenqi Fu, Hanjie Fu, Hao Ge, Hongyi Guo, Mingji Han, Li Han, Wen-  
 539 hao Hao, Xintong Hao, Qianyu He, Jerry He, Feng He, Wen Heng, Zehua Hong, Qi Hou, Liang  
 Hu, Shengding Hu, Nan Hu, Kai Hua, Qi Huang, Ziyue Huang, Hongzhi Huang, Zihao Huang,

540 Ting Huang, Wenhao Huang, Wei Jia, Bin Jia, Xiaoying Jia, Yuhua Jiang, Haobin Jiang, Ziheng  
 541 Jiang, Kaihua Jiang, Chengquan Jiang, Jianpeng Jiao, Xiaoran Jin, Xing Jin, Xunhao Lai, Zheng  
 542 Li, Xiang Li, Liyi Li, Hongkai Li, Zheng Li, Shengxian Wan, Ya Wang, Yunshui Li, Chenggang  
 543 Li, Niuniu Li, Siyu Li, Xi Li, Xiao Li, Aoyan Li, Yuntao Li, Nianning Liang, Xinnian Liang,  
 544 Haibin Lin, Weijian Lin, Ye Lin, Zhicheng Liu, Guanlin Liu, Guanlin Liu, Chenxiao Liu, Yan  
 545 Liu, Gaohong Liu, Juncuai Liu, Chundian Liu, Deyi Liu, Kaibo Liu, Siyao Liu, Qi Liu, Yongfei  
 546 Liu, Kang Liu, Gan Liu, Boyi Liu, Rui Long, Weiqiang Lou, Chenwei Lou, Xiang Luo, Yao Luo,  
 547 Caiping Lv, Heyang Lv, Bole Ma, Qianli Ma, Hongzhi Ma, Yiyuan Ma, Jin Ma, Wenchang Ma,  
 548 Tingting Ma, Chen Mao, Qiyang Min, Zhe Nan, Guanghan Ning, Jinxiang Ou, Haojie Pan, Ren-  
 549 ming Pang, Yanghua Peng, Tao Peng, Lihua Qian, Lihua Qian, Mu Qiao, Meng Qu, Cheng Ren,  
 550 Hongbin Ren, Yong Shan, Wei Shen, Ke Shen, Kai Shen, Guangming Sheng, Jinlong Shi, Wenlei  
 551 Shi, Guang Shi, Shuai Shuai Cao, Yuxin Song, Zuquan Song, Jing Su, Yifan Sun, Tao Sun, Zewei  
 552 Sun, Borui Wan, Zihan Wang, Xiaohui Wang, Xi Wang, Shuguang Wang, Jun Wang, Qinlong  
 553 Wang, Chenyuan Wang, Shuai Wang, Zihan Wang, Changbao Wang, Jiaqiang Wang, Shihang  
 554 Wang, Xuwu Wang, Zaiyuan Wang, Yuxuan Wang, Wenqi Wang, Taiqing Wang, Chengzhi Wei,  
 555 Houmin Wei, Ziyun Wei, Shufa Wei, Zheng Wu, Yonghui Wu, Yangjun Wu, Bohong Wu, Shuang  
 556 Wu, Jingqiao Wu, Ning Wu, Shuangzhi Wu, Jianmin Wu, Chenguang Xi, Fan Xia, Yuqiao Xian,  
 557 Liang Xiang, Boren Xiang, Bowen Xiao, Zhen Xiao, Xia Xiao, Yongsheng Xiao, Chao Xin,  
 558 Shulin Xin, Yuwen Xiong, Jingjing Xu, Ziwen Xu, Chenyin Xu, Jiayi Xu, Yifan Xu, Wei Xu,  
 559 Yufei Xu, Shikun Xu, Shipeng Yan, Shen Yan, Qingping Yang, Xi Yang, Tianhao Yang, Yuehang  
 560 Yang, Yuan Yang, Ximing Yang, Zeyu Yang, Guang Yang, Yifan Yang, Xuesong Yao, Bairen  
 561 Yi, Fan Yin, Jianian Yin, Ziqiang Ying, Xiangyu Yu, Hongli Yu, Song Yu, Menghan Yu, Huan  
 562 Yu, Siyu Yuan, Jun Yuan, Yutao Zeng, Tianyang Zhan, Zheng Zhang, Yun Zhang, Mofan Zhang,  
 563 Wang Zhang, Ru Zhang, Zhi Zhang, Tianqi Zhang, Xinyi Zhang, Zhixi Zhang, Sijun Zhang, Wen-  
 564 qiang Zhang, Xiangxiang Zhang, Yongtao Zhang, Yuyu Zhang, Ge Zhang, He Zhang, Yue Zhang,  
 565 Renjie Zheng, Ningxin Zheng, Zhuolin Zheng, Yaowei Zheng, Chen Zheng, Xiaoyun Zhi, Wan-  
 566 jun Zhong, Cheng Zhong, Zheng Zhong, Baoquan Zhong, Xun Zhou, Na Zhou, Huan Zhou, Hang  
 567 Zhu, Defa Zhu, Wenjia Zhu, and Lei Zuo. Seed1.5-thinking: Advancing superb reasoning models  
 568 with reinforcement learning, 2025a. URL <https://arxiv.org/abs/2504.13914>.  
 569  
 570 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared  
 571 Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri,  
 572 Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan,  
 573 Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian,  
 574 Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios  
 575 Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebbgen Guss, Alex Nichol, Alex  
 576 Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders,  
 577 Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec  
 578 Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob Mc-  
 579 Grew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large  
 580 language models trained on code, 2021. URL <https://arxiv.org/abs/2107.03374>.  
 581  
 582 Simin Chen, Xiaoning Feng, Xiaohong Han, Cong Liu, and Wei Yang. PPM: automated genera-  
 583 tion of diverse programming problems for benchmarking code generation models. *Proc. ACM*  
 584 *Softw. Eng.*, 1(FSE):1194–1215, 2024. doi: 10.1145/3643780. URL <https://doi.org/10.1145/3643780>.  
 585  
 586 Simin Chen, Pranav Pusarla, and Baishakhi Ray. Dynamic benchmarking of reasoning capabilities  
 587 in code large language models under data contamination, 2025b. URL <https://arxiv.org/abs/2503.04149>.  
 588  
 589 Yao Cheng, Jianfeng Chen, Jie Chen, Li Chen, Liyu Chen, Wentao Chen, Zhengyu Chen, Shijie  
 590 Geng, Aoyan Li, Bo Li, Bowen Li, Linyi Li, Boyi Liu, Jiaheng Liu, Kaibo Liu, Qi Liu, Shukai  
 591 Liu, Siyao Liu, Tianyi Liu, Tingkai Liu, Yongfei Liu, Rui Long, Jing Mai, Guanghan Ning, Z. Y.  
 592 Peng, Kai Shen, Jiahao Su, Jing Su, Tao Sun, Yifan Sun, Yunzhe Tao, Guoyin Wang, Siwei  
 593 Wang, Xuwu Wang, Yite Wang, Zihan Wang, Jinxiang Xia, Liang Xiang, Xia Xiao, Yongsheng  
 594 Xiao, Chenguang Xi, Shulin Xin, Jingjing Xu, Shikun Xu, Hongxia Yang, Jack Yang, Yingxiang  
 595 Yang, Jianbo Yuan, Jun Zhang, Yufeng Zhang, Yuyu Zhang, Shen Zheng, He Zhu, and Ming Zhu.  
 596 Fullstack bench: Evaluating llms as full stack coders, 2025. URL <https://arxiv.org/abs/2412.00535>.

594 Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan  
 595 Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho,  
 596 Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified, 2025.  
 597 URL <https://openai.com/index/introducing-swe-bench-verified/>.

598 Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit  
 599 Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marrs, Sam Petulla, Colin  
 600 Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-  
 601 Jiang Jiang, Krishna Haridasan, Ahmed Omran, Nikunj Saunshi, Dara Bahri, Gaurav Mishra, Eric  
 602 Chu, Toby Boyd, Brad Hekman, Aaron Parisi, Chaoyi Zhang, Kornraphop Kawintiranon, et al.  
 603 Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next  
 604 generation agentic capabilities, 2025. URL <https://arxiv.org/abs/2507.06261>.

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

608 Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo,  
 609 Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring  
 610 coding challenge competence with APPS. In Joaquin Vanschoren and Sai-Kit Yeung (eds.),  
 611 *Proceedings of the Neural Information Processing Systems Track on Datasets and Bench-  
 612 marks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL  
 613 [https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/  
 614 hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html](https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html).

615 Md. S. Hossain, Anika Tabassum, Md. Fahim Arefin, and Tarannum Shaila Zaman. Llm-pros:  
 616 Analyzing large language models' performance in competitive problem solving. In *IEEE/ACM  
 617 International Workshop on Large Language Models for Code, LLM4Code@ICSE 2025, Ottawa,  
 618 ON, Canada, May 3, 2025*, pp. 80–87. IEEE, 2025. doi: 10.1109/LLM4CODE66737.2025.00015.  
 619 URL <https://doi.org/10.1109/LLM4Code66737.2025.00015>.

620 Wenhao Hu, Jinhao Duan, Chunchen Wei, Li Zhang, Yue Zhang, and Kaidi Xu. Dynacode: A  
 621 dynamic complexity-aware code benchmark for evaluating large language models in code gen-  
 622 eration. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar  
 623 (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria,  
 624 July 27 - August 1, 2025*, pp. 21980–21997. Association for Computational Linguistics, 2025.  
 625 URL <https://aclanthology.org/2025.findings-acl.1133/>.

626 Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando  
 627 Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free  
 628 evaluation of large language models for code. In *The Thirteenth International Conference on  
 629 Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.  
 630 URL <https://openreview.net/forum?id=chfJJYC3iL>.

631 Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R.  
 632 Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The  
 633 Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria,  
 634 May 7-11, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=VTF8yNQM66>.

635 Kimi-Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen,  
 636 Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi K2: Open agentic intelligence. *arXiv  
 637 preprint arXiv:2507.20534*, 2025.

638 Jia Li, Ge Li, Xuanming Zhang, Yihong Dong, and Zhi Jin. Evocodebench: An evolving code gen-  
 639 eration benchmark aligned with real-world code repositories, 2024. URL <https://arxiv.org/abs/2404.00599>.

640 Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and  
 641 Ge Li. TACO: Topics in Algorithmic COde generation dataset, December 2023. URL <http://arxiv.org/abs/2312.14852>. arXiv:2312.14852 version: 3.

648 Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom  
 649 Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cy-  
 650 prien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl,  
 651 Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Rob-  
 652 son, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-  
 653 level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/  
 654 science.abq1158. URL <https://www.science.org/doi/abs/10.1126/science.abq1158>.  
 655

656 Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao,  
 657 Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint*  
 658 *arXiv:2412.19437*, 2024.  
 659

660 Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by  
 661 chatgpt really correct? rigorous evaluation of large language models for code generation. In  
 662 Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine  
 663 (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural*  
 664 *Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 -*  
 665 *16, 2023*, 2023. URL [http://papers.nips.cc/paper\\_files/paper/2023/hash/43e9d647cc3e4b7b5baab53f0368686-Abstract-Conference.html](http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647cc3e4b7b5baab53f0368686-Abstract-Conference.html).  
 666

667 OpenAI. GPT-4.1. <https://openai.com/index/gpt-4-1/>, a. Accessed: 2025-04-14.  
 668

669 OpenAI. GPT-4o. <https://openai.com/index/hello-gpt-4o/>, b. Accessed: 2024-  
 670 05-13.

671 OpenAI. o4-mini-high. <https://openai.com/index/introducing-o3-and-o4-mini/>, c. Accessed: 2025-04-16.  
 672

673 OpenAI, Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev, Daniel Selsam, David  
 674 Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, Jerry Tworek, Lorenz  
 675 Kuhn, Lukasz Kaiser, Mark Chen, Max Schwarzer, Mostafa Rohaninejad, Nat McAleese, o3 con-  
 676 tributors, Oleg Mürk, Rhythm Garg, Rui Shu, Szymon Sidor, Vineet Kosaraju, and Wenda Zhou.  
 677 Competitive programming with large reasoning models, 2025. URL <https://arxiv.org/abs/2502.06807>.  
 678

679 Shanghaoran Quan, Jiaxi Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren,  
 680 Bofei Gao, Yibo Miao, Yunlong Feng, et al. CodeElo: Benchmarking competition-level code  
 681 generation of llms with human-comparable Elo ratings. *arXiv preprint arXiv:2501.01257*, 2025.  
 682

683 Quan Shi, Michael Tang, Karthik Narasimhan, and Shunyu Yao. Can language models solve  
 684 olympiad programming? *arXiv preprint arXiv:2404.10952*, 2024.  
 685

686 Joshua Vendrow, Edward Vendrow, Sara Beery, and Aleksander Madry. Do large language model  
 687 benchmarks test reliability?, 2025. URL <https://arxiv.org/abs/2502.03461>.  
 688

689 Zhexu Wang, Yiping Liu, Yejie Wang, Wenyang He, Bofei Gao, Muxi Diao, Yanxu Chen, Kelin Fu,  
 690 Flood Sung, Zhilin Yang, Tianyu Liu, and Weiran Xu. Ojbench: A competition level code bench-  
 691 mark for large language models, 2025a. URL <https://arxiv.org/abs/2506.16395>.  
 692

693 Zihan Wang, Siyao Liu, Yang Sun, Hongyan Li, and Kai Shen. CodeContests+: High-quality  
 694 test case generation for competitive programming, 2025b. URL <https://arxiv.org/abs/2506.05817>.  
 695

696 Shiyi Xu, Yiwen Hu, Yingqian Min, Zhipeng Chen, Wayne Xin Zhao, and Ji-Rong Wen. Icp-  
 697 eval: Probing the frontiers of llm reasoning with competitive programming contests, 2025. URL  
 698 <https://arxiv.org/abs/2506.04894>.  
 699

700 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu,  
 701 Chang Gao, Chengan Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint*  
*arXiv:2505.09388*, 2025a.

702 Lei Yang, Renren Jin, Ling Shi, Jianxiang Peng, Yue Chen, and Deyi Xiong. Probench: Benchmark-  
 703 ing large language models in competitive programming, 2025b. URL <https://arxiv.org/abs/2502.20868>.  
 704

705 Hao Yu, Bo Shen, Dezhong Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianx-  
 706 iang Wang, and Tao Xie. Codereval: A benchmark of pragmatic code generation with generative  
 707 pre-trained models. In *Proceedings of the 46th IEEE/ACM International Conference on Software  
 708 Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pp. 37:1–37:12. ACM, 2024. doi:  
 709 10.1145/3597503.3623316. URL <https://doi.org/10.1145/3597503.3623316>.  
 710

711 Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu,  
 712 Xiaojian Zhong, Aoyan Li, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su,  
 713 Tianyu Liu, Rui Long, Kai Shen, and Liang Xiang. Multi-swe-bench: A multilingual benchmark  
 714 for issue resolving, 2025. URL <https://arxiv.org/abs/2504.02605>.

715 Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang,  
 716 Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin  
 717 Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao  
 718 Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu,  
 719 Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua  
 720 Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin,  
 721 Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong  
 722 Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran  
 723 Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen,  
 724 Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue  
 725 Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao  
 726 Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan,  
 727 Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao,  
 728 Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou  
 729 Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang,  
 730 Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu,  
 731 Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu,  
 732 Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying  
 733 Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang  
 734 Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezen Liu,  
 735 Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan  
 736 Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang  
 737 Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie  
 738 Tang. Gbm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL <https://arxiv.org/abs/2508.06471>.  
 739

740 Zihan Zheng, Zerui Cheng, Zeyu Shen, Shang Zhou, Kaiyuan Liu, Hansen He, Dongruixuan Li,  
 741 Stanley Wei, Hangyi Hao, Jianzhu Yao, Peiyao Sheng, Zixuan Wang, Wenhao Chai, Aleksandra  
 742 Korolova, Peter Henderson, Sanjeev Arora, Pramod Viswanath, Jingbo Shang, and Saining Xie.  
 743 Livecodebench pro: How do olympiad medalists judge llms in competitive programming?, 2025.  
 744 URL <https://arxiv.org/abs/2506.11928>.  
 745  
 746  
 747  
 748  
 749  
 750  
 751  
 752  
 753  
 754  
 755

756 **A EXPERIMENT SETTINGS**  
757758 The experimental platform is equipped with 3.8 GHz Intel CPUs and is isolated into several pods  
759 using Docker. The container environment is SandboxFusion (Cheng et al., 2025) with Ubuntu 20.04,  
760 where each container instance is exclusively allocated 2 cores and 4 GB of memory. Each container  
761 instance runs only one piece of code at a time. The `gcc` version used for compiling the code is  
762 9.4.0, with the C++17 standard and O2 optimization enabled.763 We use the following user prompt for the evaluation.  
764765 Please solve the following programming problem using {LANGUAGE}.  
766 Please place your final answer in a markdown code block.  
767 {STATEMENT}768 **B CATEGORY DETAILS**  
769770 The distribution of problem difficulty for each primary category is presented in Table 5, and the  
771 complete list of the primary categories and secondary categories (tags) is presented in Table 6.  
772773 **C COMPLETE PROCEDURE OF TEST CASE GENERATION**  
774775 The complete test case generation process is as follows:  
776777 1. **Writing validator.** A validator is a program used to verify whether a test case input adheres  
778 to the problem’s constraints. We first utilize the validator agent to generate the validator  
779 program, followed by manual correction of any errors.  
780 2. **Writing generator.** We employ the generator agent to create a generator program. This  
781 program is then used to produce test case inputs, which are passed to the ground truth  
782 solution to obtain the corresponding test case outputs.  
783 3. **Writing checker and interactor.** We utilize checker and interactor agents to generate the  
784 respective programs. A checker program is essential for problems that accept multiple valid  
785 solutions, while an interactor program is required for interactive problems. Subsequently,  
786 these undergo manual review and error correction.  
787 4. **Human expert augmentation.** Human experts supplement the machine-generated test  
788 cases, adding new cases until a 100% TNR is achieved on the collected solution set.  
789 5. **Elite team audit.** Finally, our elite team conducts a comprehensive review of each problem.  
790 This process includes adding corner cases, rejecting unqualified samples, and specifically  
791 authoring incorrect or inefficient solutions to re-verify the coverage of the test cases.  
792793 **D SOURCE AND COPYRIGHT DETAILS**  
794795 The complete list of the contest sources of AetherCode v1 is presented in Table 7.  
796797 Some of the problems have clear copyright holders and licenses, including:  
798801 • **IOI.** Copyright held by the IOI General Assembly; released under the CC BY License.  
802 • **JOI.** Copyright held by The Japanese Committee of International Olympiad in Informatics;  
803 released under the CC BY-SA 4.0 License.  
804 • **USACO.** Copyright held by USACO; released under the CC BY-NC-SA 4.0 License.  
805 • **NOI (China).** Copyright held by the China Computer Federation; released under the CC  
806 BY-NC 4.0 License.  
807808 For some problems, the authorization or copyright status is currently unverifiable. We remain com-  
809 mitted to removing any potentially infringing problems upon the request of the copyright holders.

810 E DETAILS OF FAILURE ANALYSIS  
811812 The distribution of failure reasons (by judging verdicts) across evaluated models is presented in  
813 Table 8.814 We performed a qualitative analysis of the causes of errors in o4-mini-high. The primary error  
815 categories and their corresponding cases are listed below. The problems and responses for these  
816 cases are provided in the supplementary material.817  
818 

- cases/acknowledge\_inability: The model admits its inability to complete the  
819 problem, outputting: “I’m sorry, but I can’t get to a working solution in the time I have.”
- cases/corner\_case: The model identified key properties but failed to properly handle  
820 all corner cases.
- cases/implementation\_error: The model encountered an error during code im-  
821 plementation, specifically failing to close parentheses.
- cases/incorrect\_logic: The model failed to employ the correct algorithmic logic.
- cases/inefficient: The model used an inefficient algorithm. In this example, the  
822 model correctly calculated the algorithm’s time complexity but failed to realize that an  
823  $O(n^3)$  algorithm is typically unable to handle a data scale of  $n = 5000$  within one second.

  
824830 F LLM USAGE  
831832 In this work, we utilized LLMs to facilitate writing on tasks such as text refinement, translation, and  
833 searching for related literature. Furthermore, Vision-Language Models (VLMs) were employed for  
834 the generation of illustrations.835 Table 5: Distribution of problem difficulty for each Category in AetherCode v1 (2401-2505).  
836837  
838 

Category	Easy	Medium	Hard	Extreme
Basic Algorithms	43.11%	27.11%	25.78%	4.00%
Common Techniques	30.61%	42.18%	21.77%	5.44%
Computational Geometry	16.67%	30.56%	47.22%	5.56%
Data Structures	23.33%	33.33%	37.50%	5.83%
Dynamic Programming	20.91%	30.91%	45.45%	2.73%
Graph Theory	21.88%	31.25%	37.50%	9.38%
Mathematics	25.00%	32.29%	36.46%	6.25%
Search	28.00%	32.00%	36.00%	4.00%
Strings	38.46%	15.38%	38.46%	7.69%
Tree Problems	16.67%	25.00%	50.00%	8.33%

  
839

864  
865  
866  
867  
868  
869  
870  
871

Table 6: Category division and detailed tag distribution of AetherCode.

Category	Tags
<b>Algorithm Basics</b>	Enumeration, Simulation, Recursion, Greedy, Sorting, Divide and Conquer, Binary Search, Doubling, Recurrence
<b>Search</b>	DFS, BFS, Bidirectional Search, Heuristic Search, A*, Iterative Deepening Search, IDA*, Dancing Links
<b>Dynamic Programming</b>	Basic DP, Memorization Search, Knapsack DP, Range DP, DP on DAGs, Tree DP, Bitmask DP, Digit DP, Plug DP, Counting DP, Dynamic DP, Probability DP, DP Optimization
<b>Strings</b>	String Matching, String Hashing, Trie, Palindrome Automation, Prefix Function, Z-function, Automation, AC Automation, Suffix Array, Suffix Automation, Suffix Balanced Tree, Generalized Suffix Automation, Suffix Tree, Manacher’s Algorithm, KMP Algorithm, Sequence Automation, Minimal Representation, Lyndon Factorization, Main-Lorentz Algorithm
<b>Mathematics</b>	Number Theory, Linear Algebra, Linear Programming, Abstract Algebra, Probability Theory, Game Theory, Young Matrix, Inclusion-Exclusion Principle, Combinatorics, Polynomials
<b>Data Structures</b>	Stack, Queue, Linked List, Hash Table, Disjoint Set Union, Heap, Block Structure, Monotonic Queue, ST Table, Binary Indexed Tree, Segment Tree, Balanced Tree, Binary Tree & Balanced Tree, Block Decomposition, Persistent Data Structures, Tree-in-Tree, K-D Tree, Cartesian Tree, Huffman Tree, STL-based Data Structure
<b>Graph Theory</b>	Matrix-Tree Theorem, Directed Acyclic Graph, Topological Sort, Minimum Spanning Tree, Minimum Diameter Spanning Tree, Minimum Tree Spanning, Connectivity, Shortest Path, 2-SAT, Difference Constraints, Hamiltonian Graph, Modular Shortest Path, Graph Coloring, Eulerian Graph, Dominating Tree, Bipartite Graph, Prüfer Sequence, Planar Graph, Chordal Graph, Network Flow, Graph Matching, Random Walk on Graphs, LGV Lemma, Strongly Connected Components
<b>Computational Geometry</b>	Euclidean Distance, Manhattan Distance, Chebyshev Distance, Pick’s Theorem, Triangulation, Convex Hull, Sweep Line, Rotating Calipers, Half-Plane Intersection, Closest Pair of Points, Random Increment Method, Reflection Transformation, Misc. CG
<b>Common Techniques</b>	Discretization, Two Pointer Technique, Prefix Sum & Difference, Fractional Programming, Randomization, Hanging Line Method, Binary Thinking, Pattern Recognition, Gray Code, Expression Evaluation, Construction, Properties of Bitwise Operations, Conjecture of Conclusions, Interactive Problems, Meet in Middle, Ad-hoc, Uncertainty Algorithms, Square Root Decomposition
<b>Problems on Trees</b>	LCA, DSU on Tree, Divide and Conquer on Points, Block Decomposition on Tree, Heavy-Light Decomposition, Chain Decomposition, Tree Diameter and Centroid, LCT

900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917

Table 7: Curated Contest Source of AetherCode v1 (2401-2505).

918	Competition Name	Category	Date
919	Croatian Open Competition in Informatics 2023/2024 Contest #3	Croatian OI	2024/1/13
920	USACO 2024 January Contest (Platinum)	USACO Platinum	2024/1/26
921	The 2023-2024 ICPC Southwestern Europe Regional Contest	ICPC Regional Contests	2024/1/28
922	Croatian Open Competition in Informatics 2023/2024 Contest #4	Croatian OI	2024/2/10
923	USACO 2024 February Contest (Platinum)	USACO Platinum	2024/2/16
924	USACO 2024 US Open Contest (Platinum)	USACO Platinum	2024/3/15
925	Singapore National Olympiad in Informatics 2024 Final Contest	NOI (SG)	2024/3/16
926	Croatian Open Competition in Informatics 2023/2024 Contest #5	Croatian OI	2024/3/16
927	The 2024 ICPC Latin America Championship	ICPC Regional Championships/Finals	2024/3/17
928	The 2024 ICPC Europe Championship	ICPC Regional Championships/Finals	2024/3/24
929	The 2024 British Informatics Olympiad Final	British OI	2024/4/6
930	Baltic Olympiad in Informatics 2024 Day 1	Baltic OI	2024/5/4
931	Baltic Olympiad in Informatics 2024 Day 2	Baltic OI	2024/5/5
932	Asia-Pacific Informatics Olympiad 2024 (APIO 2024)	APIO	2024/5/18
933	The 2024 ICPC North America Championship	ICPC Regional Championships/Finals	2024/5/27
934	Central European Olympiad in Informatics 2024 Day 1 (CEOI 2024 Day 1)	Central European OI	2024/6/25
935	Central European Olympiad in Informatics 2024 Day 2 (CEOI 2024 Day 2)	Central European OI	2024/6/27
936	China National Olympiad in Informatics 2024 Day 1	NOI	2024/7/18
937	China National Olympiad in Informatics 2024 Day 2	NOI	2024/7/20
938	European Girls' Olympiad in Informatics 2024 Day 1	European Girl's OI	2024/7/23
939	European Girls' Olympiad in Informatics 2024 Day 2	European Girl's OI	2024/7/25
940	International Olympiad in Informatics 2024 Day 1	IOI	2024/9/3
941	International Olympiad in Informatics 2024 Day 2	IOI	2024/9/5
942	The 2024 ICPC World Finals Astana	ICPC World Finals	2024/9/19
943	The 2024 ICPC Kunming Invitational Contest	ICPC Regional Contests	2024/9/28
944	The 2024 Nordic Collegiate Programming Contest	NCPC	2024/10/5
945	Croatian Open Competition in Informatics 2024/2025 Contest #1	Croatian OI	2024/10/5
946	CCPC 2024 Harbin Site	CCPC	2024/10/26
947	The 2024 ICPC Asia Chengdu Regional Contest	ICPC Regional Contests	2024/10/27
948	The 2024 ICPC Asia Nanjing Regional Contest	ICPC Regional Contests	2024/11/3
949	Croatian Open Competition in Informatics 2024/2025 Contest #2	Croatian OI	2024/11/9
950	2024-2025 ICPC Latin American Regional Programming Contest	ICPC Regional Championships/Finals	2024/11/9
951	2024 Rocky Mountain Regional Contest	ICPC Regional Contests	2024/11/9
952	2024 North Central NA Regional Contest	ICPC Regional Contests	2024/11/9
953	2024 Mid-Central USA Programming Contest	ICPC Regional Contests	2024/11/9
954	CCPC 2024 Chongqing Site	CCPC	2024/11/10
955	The 2024 ICPC Greater NY Regional Contest	ICPC Regional Contests	2024/11/10
956	The 2024 ICPC Asia Hangzhou Regional Contest	ICPC Regional Contests	2024/11/10
957	CCPC 2024 Jinan Site	CCPC	2024/11/16
958	The 2024 ICPC Pacific Northwest Regional Contest (Div. 1)	ICPC Regional Contests	2024/11/16
959	The 2024 ICPC Pacific Northwest Regional Contest (Div. 2)	ICPC Regional Contests	2024/11/16
960	ICPC NA South Division 2024 - Division 2	ICPC Regional Contests	2024/11/16
961	ICPC NA South Division 2024 - Division 1	ICPC Regional Contests	2024/11/16
962	The 2024 ICPC Southern California Regional Contest	ICPC Regional Contests	2024/11/16
963	The 2024 ICPC Southeastern Europe Regional Contest (SEERC 2024)	ICPC Regional Contests	2024/11/17
964	The 2024 ICPC Asia Shanghai Regional Contest	ICPC Regional Contests	2024/11/17
965	The 2024 ICPC Asia Seoul Regional Contest	ICPC Regional Contests	2024/11/23
966	The 2024 ICPC Northwestern Europe Regional Contest (NWERC 2024)	ICPC Regional Contests	2024/11/24
967	The 2024 ICPC Asia Shenyang Regional Contest	ICPC Regional Contests	2024/11/24
968	Romanian Master of Informatics 2024 Day 1	Romanian OI	2024/11/28
969	Romanian Master of Informatics 2024 Day 2	Romanian OI	2024/11/29
970	The 2024 ICPC Asia Kunming Regional Contest	ICPC Regional Contests	2024/12/1
971	Croatian Open Competition in Informatics 2024/2025 Contest #3	Croatian OI	2024/12/12
972	USACO 2024 December Contest (Platinum)	USACO Platinum	2024/12/13
973	The 2024 ICPC Northern Eurasia Finals	ICPC Regional Championships/Finals	2024/12/15
974	The 2024 ICPC Central Europe Regional Contest	ICPC Regional Contests	2024/12/15
975	CCPC 2024 Zhengzhou Site	CCPC	2024/12/21
976	The 2024 ICPC Asia Yokohama Regional Contest	ICPC Regional Contests	2024/12/22
977	The 2024 ICPC Asia Hong Kong Regional Contest	ICPC Regional Contests	2024/12/22
978	The 2024 ICPC Asia East Continent Final Contest	ICPC Regional Championships/Finals	2024/12/28
979	USACO 2025 January Contest (Platinum)	USACO Platinum	2025/1/24
980	Croatian Open Competition in Informatics 2024/2025 Contest #4	Croatian OI	2025/1/25
981	The 24th Japanese Olympiad in Informatics Final Round (JOI 2024/2025)	Japanese OI	2025/2/2
982	Croatian Open Competition in Informatics 2024/2025 Contest #5	Croatian OI	2025/2/15
983	USACO 2025 February Contest (Platinum)	USACO Platinum	2025/2/21
984	The 2025 ICPC Europe Championship	ICPC Regional Championships/Finals	2025/3/2
985	2025 ICPC Asia West Finals	ICPC Regional Championships/Finals	2025/3/7
986	The 2025 ICPC Latin America Championship	ICPC Regional Championships/Finals	2025/3/16
987	USACO 2025 US Open Contest (Platinum)	USACO Platinum	2025/3/21
988	Singapore National Olympiad in Informatics 2025 Final Contest	NOI (SG)	2025/3/22
989	The 2025 British Informatics Olympiad Final	British OI	2025/4/12
990	Baltic Olympiad in Informatics 2025 Day 1	Baltic OI	2025/4/26
991	Baltic Olympiad in Informatics 2025 Day 2	Baltic OI	2025/4/27
992	The 2025 ICPC China Zhejiang Province Programming Contest (22nd)	ICPC Regional Contests	2025/5/10
993	CCPC Final 2024	CCPC Final	2025/5/11
994	Asia-Pacific Informatics Olympiad 2025 (APIO 2025)	APIO	2025/5/17

Competition Name	Category	Date
The 2025 ICPC Asia Wuhan Invitational Contest	ICPC Regional Contests	2025/5/17
The 2025 ICPC North America Championship	ICPC Regional Championships/Finals	2025/5/26

Table 8: Distribution of failure reasons across evaluated models (% of total errors).

Model	Wrong Answer	Time Limit	Runtime Error	Compile Error
<i>Reasoning Models</i>				
o4-mini-high	86.0	6.1	0.3	7.6
Gemini-2.5-Pro	76.3	18.1	0.1	5.4
Seed-1.6-thinking-0715	79.1	15.2	0.1	5.6
DeepSeek-R1-0528	77.1	11.1	0.1	11.7
Qwen-3-235B-A22B-thinking	81.3	12.3	0.0	6.4
Gemini-2.5-Flash	79.7	11.4	0.1	8.9
GLM-4.5	71.0	10.5	0.0	18.5
Qwen-3-235B-A22B	77.8	12.0	0.1	10.1
Qwen-3-32B	77.7	13.8	0.1	8.5
Claude-Sonnet-4.5-thinking	45.8	51.7	0.0	2.5
Claude-4-Opus-thinking	48.2	48.3	0.0	3.5
Claude-4-Sonnet-thinking	50.8	45.8	0.0	3.4
Qwen-3-8B	69.2	9.1	0.1	21.7
<i>Non-Reasoning Models</i>				
GPT-4.1	79.3	12.5	0.1	8.1
Kimi-K2	77.0	7.2	0.0	15.7
DeepSeek-V3	82.8	9.2	0.1	7.8
Qwen-3-Coder-480B-A35B-Instruct	78.9	15.3	0.1	5.8
Claude-4-Sonnet	65.2	30.7	0.0	4.0
GPT-4o	72.1	8.5	0.1	19.3

## G EXAMPLE PROBLEMS

### Example 1

**Source:** The 2024 ICPC World Finals Astana

**Title:** The Silk Road . . . with Robots!

**Time limit:** 5 seconds

Parts of the ancient silk road passed through southern Kazakhstan. You've been fantasizing about a modern silk road, which has its own special features. Along your fantasy road are robots as well as stores holding stashes of tenges (the national currency of Kazakhstan). If a robot moves to a location with a store, the robot collects all that store's tenges for you.

The cost of moving a robot is 1 tenge for every meter moved. So the amount of profit from moving a robot to a store is the number of tenges held by the store minus the number of meters the robot has moved to reach the store.

Consider this scenario, which stretches over several days. Initially, the road is empty, with no robots or stores. Every day, either a new robot or a new store is placed on an unoccupied location along the road. Immediately before that, each existing store on the road is restocked with tenges so that its total amount is the same as it was when it was first placed on the road, and each robot is returned to its original starting location.

For each day, you need to determine the maximum amount of profit that could be gained by moving robots to collect tenges from the stores. Note that no two robots start in the same location, but they may occupy the same location as they move. Each store can be emptied of

1026 its tenges only once during a single day.  
 1027  
 1028  
 1029 **Input**  
 1030 The first line contains an integer  $n$  ( $1 \leq n \leq 2 \cdot 10^5$ ), the number of days. This is followed by  
 1031  $n$  lines, where the  $i$ -th line starts with an integer  $t_i$ , which is equal to 1 if a new robot is added  
 1032 on day  $i$ , or is equal to 2 if a new store is added that day.  
 1033 If  $t_i = 1$ , the line contains another integer  $x_i$  ( $0 \leq x_i \leq 10^8$ ), denoting the location of the new  
 1034 robot.  
 1035 If  $t_i = 2$ , the line contains another integer  $x_i$  ( $0 \leq x_i \leq 10^8$ ) denoting the location of the new  
 1036 store, followed by another integer  $c_i$  ( $0 \leq c_i \leq 10^8$ ), denoting the number of tenges at the  
 1037 store.  
 1038 All the given locations are distinct.  
 1039  
 1040 **Output**  
 1041 Output  $n$  integers, the maximum profit you can make after each day.  
 1042  
 1043 **# Sample Input:**  
 1044 6  
 1045 1 20  
 1046 2 15 15  
 1047 2 40 50  
 1048 1 50  
 1049 2 80 20  
 1050 2 70 30  
 1051  
 1052 **# Sample Output:**  
 1053 0  
 1054 10  
 1055 35  
 1056 50  
 1057 50  
 1058 60

## Example 2

1059 **Source:** The 2024 ICPC Asia East Continent Final Contest  
 1060 **Title:** Boolean Function Reconstruction  
 1061 **Input file:** standard input  
 1062 **Output file:** standard output  
 1063 **Time limit:** 4 seconds  
 1064 **Memory limit:** 1024 megabyte  
 1065  
 1066 Given the truth table of a boolean function with  $n$  boolean variables as input, construct an  
 1067 expression that satisfies this function. In the expression, you are only allowed to use the  
 1068 logical and ( $\&$ ) and logical or ( $|$ ) operators. Specifically, a truth table of a boolean function  
 1069 with  $n$  boolean variables gives all the  $2^n$  outputs corresponding to the possible values of  $n$   
 1070 input variables. A boolean expression  $\langle \text{expr} \rangle$  has the following forms:  
 1071  
 1072 

- T, F: Represents True and False.
- a, b, ..., z: Represents one of the variables. The  $i$ -th variable is represented by the  $i$ -th  
 1073 lowercase letter in alphabetical order.
- $\langle \text{expr} \rangle \& \langle \text{expr} \rangle$ : Represents the logical and operation applied to the results of two  
 1074 expressions.
- $\langle \text{expr} \rangle | \langle \text{expr} \rangle$ : Represents the logical or operation applied to the results of two  
 1075 expressions.

 1076  
 1077  
 1078  
 1079

1080  
1081  
1082

The logical and operation and the logical or operation are defined as two boolean functions below that take two boolean values.

1083  
1084  
1085  
1086  
1087  
1088

$x_1$	$x_2$	$x_1 \& x_2$	$x_1   x_2$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

1089  
1090  
1091  
1092

Determine whether an expression exists that satisfies the conditions. If such an expression exists, ensure that the number of binary operators ( $\&$  and  $|$ ) does not exceed  $2^{n-1} + 10$ , and the depth of parentheses nesting does not exceed 100 layers. It can be proven that if a solution exists, there is always one that meets the constraints of the problem.

1093  
1094  
1095

### Input

The input consists of multiple test cases. The first line contains an integer  $T$  ( $1 \leq T \leq 2^{16}$ ), the number of test cases. For each test case, there are two lines:

1096  
1097  
1098  
1099

- The first line contains an integer  $n$  ( $1 \leq n \leq 15$ ).
- The second line contains a binary string  $s$  with length  $2^n$ , indicating the truth table of the given function.

1100  
1101  
1102  
1103  
1104

To interpret the input binary string, suppose the  $i$ -th variable has a value of  $x_i$ . Then, the corresponding function value,  $f(x_1, x_2, \dots, x_n)$ , is equal to the  $(\sum_{i=1}^n x_i \cdot 2^{i-1} + 1)$ -th bit of the string  $s$ .

It is guaranteed that the sum of  $2^{2n}$  over all test cases will not exceed  $2^{30}$ .

1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112

### Output

For each test case:

- Output Yes or No on the first line to indicate whether an expression satisfying the conditions exists.
- If an expression exists, output the expression on the second line. The expression must strictly adhere to the format given in the problem description, without adding or omitting parentheses, and without adding extra spaces.

1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129

### Example

standard input	standard output
7	Yes
2	(a&b)
0001	Yes
2	(a b)
0111	Yes
2	T
1111	Yes
3	((a&(b c)) (b&c))
00010111	No
1	Yes
10	a
2	Yes
0101	(a&(b&(c&(d&e))))
5	
0001	

1130  
1131  
1132  
1133

### Note

Below is the truth table interpretation for the fourth sample.

1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143

$x_3$	$x_2$	$x_1$	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187