Acta Univ. Sapientiae, Informatica, **16, 2** (2024) 255–285

DOI: 10.47745/ausi-2024-0014

The Sapientia ECN AI Baseline Index: Benchmarking Large Language Models Against Student Performance in Competitive Programming

Zoltán KÁTAI
Sapientia Hungarian University of Transylvania
Târgu Mureş, Romania
katai_zoltan@ms.sapientia.ro
0000-0003-2343-3629

David ICLANZAN
Sapientia Hungarian University of Transylvania
Târgu Mureş, Romania
iclanzan@ms.sapientia.ro
0000-0003-2587-9106

Abstract.

We introduce the Sapientia ECN AI Baseline Index, a benchmark for evaluating the fundamental problem-solving capabilities of publicly available, state-of-the-art Large Language Models (LLMs) in competitive programming. Using basic prompting techniques on problems from the annual Sapientia Efficiency Challenge Networking (ECN) competition, we assess LLMs' baseline performance, deliberately excluding more advanced enhancements like agentic systems or external knowledge retrieval.

Our initial study compares LLM results with those of student teams from the ECN 2023 competition, analyzing both the number and types of problems solved, as well as score distributions. By providing a consistent, longitudinal measure, the ECN AI Baseline Index aims to track AI baseline capability advancement in complex problem-solving domains and offers insights into the evolving strengths and limitations of LLMs relative to peak and median student expertise.

Key words and phrases: competitive programming, AI program code generation

1 Introduction

In recent years, the rapid advancement of artificial intelligence has sparked new possibilities in evaluating programming competition difficulty and participant skill levels. Programming contests, including globally recognized events like the International Collegiate Programming Contest (ICPC) and Google Code Jam, have long served as arenas for coders to test their problem-solving and algorithmic thinking. These competitions demand not only proficiency in coding but also the ability to devise efficient algorithms under strict time constraints. Similarly, the Sapientia-ECN (Efficiency Challenge Networking) programming contest, organized by the Mathematics and Informatics Department of Sapientia Hungarian University of Transylvania, has established itself as a significant event in the region, adhering to ACM-ICPC standards and attracting high school and university teams for over 15 years. The competition provides a unique and challenging environment, allowing young programmers to showcase their skills in a structured and competitive setting.

Determining an appropriate level of problem difficulty is essential to the success of any programming competition, as it influences participant engagement, learning outcomes, and the overall fairness of the event. A well-calibrated difficulty range ensures that the contest remains accessible to participants of diverse skill levels, fostering broader participation while maintaining a high standard. Balancing easy and challenging problems not only sustains motivation among participants but also enhances the educational impact of the competition, enabling students to develop stronger problem-solving and algorithmic skills. This balance is crucial for contests aimed at educational growth, as it prevents problems from being too trivial or prohibitively complex, thereby enhancing participants' sense of achievement and satisfaction.

The increasing sophistication of large language models (LLMs) has introduced a new method for evaluating problem complexity and solution effectiveness in programming competitions. State-of-the-art language models, such as those based on OpenAI's latest offerings, now serve as a practical benchmark for assessing problem difficulty by simulating solutions generated with basic prompting techniques. These models can provide a stable, consistent baseline for comparing AI performance against that of human participants, offering organizers a quantitative approach to evaluate the complexity of contest problems. Furthermore, the qualitative aspects of AI performance are increasingly relevant, as they allow us to delve into not just the quantity of successfully solved problems but also the types of problems that different models and human participants can address. By examining the distribution of scores achieved by human competitors alongside AI results, we can gain a nuanced understanding of how AI models handle various problem types compared to human teams, thereby offering a holistic view of AI's current strengths and limitations.

This paper introduces the Sapientia ECN AI Baseline Index (EAII), a novel metric that systematically evaluates the capability of publicly accessible LLMs in solving algorithmic challenges typical of programming competitions. By applying this index to the 2023 Sapientia-ECN contest, we aim to establish an accessible, replicable baseline for AI performance, focusing on standard, straightforward prompting methods that any user can implement. Through this comparative analysis, which includes both quantitative and qualitative evaluations, we assess the problem-solving abilities of current LLMs, their limitations, and the variety of problems they can tackle effectively. The EAII thus serves as a longitudinal tool, designed not only to track the evolution of AI capabilities in competitive programming but also to offer a unique perspective on the types of algorithmic challenges that remain difficult for AI, relative to human performance, as models continue to advance in complex problem-solving domains.

2 Background

Program synthesis aims to automate the coding process by generating programs that satisfy user-specified intents, a goal often considered the "holy grail" of computer science [1], [2]. Achieving this would significantly enhance programmer productivity and broaden access to programming. Automatically creating programs from high-level descriptions has long been a central challenge in computer science.

Developing AI systems capable of solving unforeseen problems by generating code from descriptions is multifaceted, advancing our understanding of problem solving and reasoning [3]. Solving competitive programming problems is a crucial step in this field. It requires understanding complex natural language descriptions, reasoning about novel problems, mastering diverse algorithms and data structures, and precisely implementing extensive solutions [4]. Competitive programming, which attracts hundreds of thousands of participants worldwide, provides robust benchmarks for evaluating intelligence [5].

2.1 State-of-the-Art Models for Competitive Programming

Large language models (LLMs) have demonstrated remarkable reasoning capabilities, though debates about their limitations and data contamination issues persist [6]. Recent studies focus on evaluating LLMs' abilities to solve expert-crafted, competition-level programming problems, which demand deep understanding and robust reasoning skills [6]. For example, the paper by Coignion [7] evaluates the performance of LLM-generated code on Leetcode, a popular platform for coding challenges. This research compares 18 different LLMs, analyzing factors such as model temperature and success rates. The findings indicate that LLMs can produce code with performance comparable to that of human-crafted solutions, suggesting that LLMs are capable of handling competitive programming tasks effectively [7].

Recent advancements of state-of-the-art models have led to remarkable progress in machine assisted solving of competitive programming problems[8], with models such as AlphaCode, AlphaCode 2, and OpenAI's o1 achieving significant milestones. These systems demonstrate AI's growing capability to tackle complex coding challenges, with each model offering distinct approaches and achievements in the field.

DeepMind's initial entry into this domain, AlphaCode, established a foundation for AI systems in competitive programming by achieving rankings in the top 54.3% of participants in simulated Codeforces competitions [4]. The system's success stems from its innovative methodology of generating millions of diverse code submissions through transformer-based networks, followed by sophisticated filtering and clustering processes that select up to ten submissions for final evaluation. This approach marked a significant step forward in replicating human-like problem-solving capabilities in programming contexts.

The subsequent development of AlphaCode 2 [9] represented a substantial advancement over its predecessor. This improved system demonstrates performance levels surpassing approximately 85% of human programmers in competitive programming scenarios, positioning it between the 'Expert' and 'Candidate Master' categories on Codeforces [10]. AlphaCode 2's enhanced capabilities are attributed to several key improvements, including a more sophisticated training methodology utilizing an expanded dataset of programming challenges. The system achieves nearly double the problem-solving efficiency of its predecessor, successfully resolving approximately 43% of problems within ten attempts, compared to the original AlphaCode's 25% success rate.

OpenAI's newest model named o1 [11] seems to have has pushed the boundaries even further [12]. It is reported to achieve rankings in the 89th percentile on Codeforces [11]. The o1 model's success is rooted in its deliberate, reasoning-centered approach to problem-solving, moving beyond the rapid response mechanisms characteristic of earlier AI systems. This model distinguishes itself through its ability to refine strategies, identify potential errors, and explore alternative methods to improve accuracy. The training methodology emphasizes deep analytical thinking, enabling the model to develop sophisticated problem-solving strategies that more closely mirror human cognitive processes. Compared to its predecessor GPT-4, the o1 model demonstrates notably enhanced performance in tasks requiring precise reasoning and stepwise analysis, particularly in advanced problem domains.

These developments collectively represent a significant evolution in AI's capability to engage in competitive programming, with each successive model demonstrating increasingly sophisticated approaches to problem-solving and higher levels of performance in competitive scenarios. The progression from AlphaCode to Alpha-

Code 2 and the introduction of the o1 model illustrate the rapid pace of advancement in this field, suggesting promising directions for future developments in AI-driven programming solutions.

3 Sapientia-ECN: a challenging programming competition

The Sapientia-ECN (Efficiency, Challenge, Networking) programming contest, organized by the Mathematics and Informatics Department of the Sapientia Hungarian University of Transylvania, has been a prestigious event for more than 15 years, targeting high school and university teams in the region. This competition follows the ACM-ICPC (International Collegiate Programming Contest) format, providing an excellent opportunity for young programmers to challenge their skills and compete in a rigorous, yet rewarding environment.

In line with the ACM-ICPC style, the Sapientia-ECN competition features:

- Teams of Three: Each team consists of three members who collaborate to solve problems. This encourages teamwork and effective communication.
- Single Computer: Teams are provided with one computer, simulating a real-world scenario where resources are limited and must be managed efficiently.
- English Language Problem Set: The problem set, written in English, ensures that participants can compete on an international level and gain experience with globally
- Five-Hour Duration: Teams have five hours to solve as many problems as they can. This tests their ability to perform under pressure and manage their time effectively.

The problems are designed to be of medium difficulty, striking a balance that challenges participants while remaining accessible to a wide range of skill levels. Solutions are evaluated based on correctness and efficiency, with no partial credits awarded for incomplete or partially correct solutions. Solutions must be completely correct to earn points.

To recognize the efforts of participants at different educational stages, the competition awards prizes separately for high school and university teams. This ensures a fair comparison and encourages participation from younger students, fostering an early interest in computer science and programming.

The mission of the Sapientia-ECN competition goes beyond just being a test of programming ability. It provides participants with:

- Experience with ACM-ICPC Standards: By following the ACM-ICPC format, participants gain familiarity with one of the most prestigious programming competitions worldwide, preparing them for future contests at higher levels.
- Networking Opportunities: The event brings together young programmers from various institutions, promoting networking and the exchange of ideas.
- Skill Development: The problems are crafted to improve algorithmic thinking, problem-solving skills, and programming proficiency, essential traits for future careers in technology and research.

In Conclusion the Sapientia-ECN programming competition stands out as a significant event in the regional academic calendar. It not only fosters a competitive spirit among young programmers but also prepares them for larger stages like the ACM-ICPC. By participating, students gain invaluable experience, develop critical skills, and become part of a vibrant community of problem solvers.

4 Annual AI Performance Evaluation

The ECN AI Baseline Index (EAII) is constructed as a relative performance index that compares the easily and consistently achievable performance of AI systems in competitive programming contests against the peak and median performances of student teams. Each year, the Index uses publicly available state-of-the-art models, accessible to any user, ensuring the performance is replicable without advanced configurations, techniques or custom system setups.

4.1 Design of an Achievable Baseline

EAII is not intended to represent the highest possible AI performance with state-of-the-art techniques like retrieval-augmented generation (RAG) or agentic systems. Instead, it focuses on a baseline score, one that any user could obtain using simple and repeatable methods. In each evaluation cycle, the chosen models are used "as-is", with basic prompting. This establishes a fair and comparable standard over time, ensuring that advancements in the metric reflect changes in general AI accessibility and capability, rather than specialized, advanced techniques or very resource-intensive processes.

The prompt used to assess the AI models is minimal yet effective, constructed to solicit a competent solution without additional guidance:

As an expert competitive programmer, write a very efficient Python 3 solution for the given problem. Read inputs from standard input and write outputs to standard output.

This prompt is followed by the problem statement as provided in the ECN contest. By using a simple and consistent prompt, we ensure that the AI's performance is based on its inherent capabilities without the influence of intricate prompting strategies and prompt engineering.

4.2 Limiting AI Solution Attempts for Fair Comparisons

AI systems can generate code rapidly and could potentially generate a very large number of solutions until one succeeds. To mitigate this advantage we implement a controlled evaluation procedure comprising two stages.

Local testing: The solution provided by the AI is first tested locally using the example inputs and outputs provided in the problem statement. If the solution produces errors, such as runtime exceptions or incorrect results, a feedback loop is created in the form of the error messages or observed incorrect behavior. If the proposed solution involves precomputation, a technique commonly used in competitive programming, the LLM is asked to separate the precomputation script and indicate where the results should be placed in the final solution. The AI is allowed a maximum of three rounds of error correction or enhancements at this stage.

Contest evaluation system: If the solution passes the local test, it is then submitted to the automatic grading system of the contest. If the solution fails the system's checks, one final feedback loop is provided, where the AI is informed of the failed test cases, along with standard error messages such as "Time Limit Exceeded" or "Wrong Answer". Only one round of feedback is allowed at this stage, limiting the AI's optimization opportunities to maintain comparability with human competitors.

By constraining the number of feedback rounds and aligning the evaluation process with the standard contest conditions, we aim to limit the AI's potential advantage derived from rapid iteration and multiple attempts. This approach ensures that the AI's performance is measured under conditions comparable to those experienced by human teams, who also have limited opportunities to debug and resubmit solutions within the contest time frame.

4.3 Baseline AI Performance

PT_{AI} denotes the total number of problems the AI successfully solves according to the conditions detailed above. A solution is considered successful only if it passes all test cases in the automated judging system of the contest. A direct relative comparison can be used to evaluate AI performance in relation to student performance.

Relative Performance to Median (RPM):

$$\text{RPM} = \frac{PT_{\text{AI}} - \text{Median}(PT_{\text{students}})}{\text{Median}(PT_{\text{students}})} \times 100 \tag{1}$$

where:

- PT_{students} is the set of total problems solved by each student team.
- $\text{Median}(PT_{\text{students}})$ is the median number of problems solved by student teams.

Relative Performance to Peak (RPP):

$$\text{RPP} = \frac{PT_{\text{AI}} - \text{Peak}(PT_{\text{students}})}{\text{Peak}(PT_{\text{students}})} \times 100 \tag{2}$$

where $\text{Peak}(PT_{\text{students}})$ is the maximum number of problems solved by any student team.

These metrics provide insights into how AI performance compares with both median and peak student performance, offering a multifaceted analysis of its capabilities in relation to different benchmarks.

4.4 EAII Definition

In addition to RPM and RPP, which reflect AI's position relative to specific student performance points, the ECN AI Baseline Index (EAII) is designed to offer a comprehensive assessment by considering both median and peak student performances. The EAII integrates these perspectives to provide a single, unifying metric:

$$\text{EAII} = \frac{PT_{\text{AI}} - \text{Median}(PT_{\text{students}}) + 1}{\text{Peak}(PT_{\text{students}}) - \text{Median}(PT_{\text{students}}) + 1} \times 100, \tag{3}$$

The EAII measures the AI's performance relative to the range between the median and peak student performances, expressed as a percentage. Laplacian smoothing ensures that the denominator is not zero in cases of uniform performance between teams.

4.5 Characteristics of the Metric

EAII provides a normalized scale, being a dimensionless quantity that normalizes the AI's performance between the median and peak student performances. This normalization facilitates comparison across different contests and time periods, even when the number and difficulty of problems might vary. As a relative performance indicator, an EAII of 0% indicates that the AI's performance is equivalent to the median student team, while an EAII of 100% signifies parity with the top-performing student team. Values exceeding 100% imply that the AI outperformed the best student team, whereas negative values indicate performance below the median.

The EAII metric becomes more sensitive when student performance is tightly clustered. When the top and median teams score similarly, the performance deviation between the student teams is low. Consequently, each problem the AI solves or fails to solve causes a larger shift in its EAII score. This is because the metric normalizes AI performance against the spread of student scores. With a narrow spread, even small changes in AI performance result in dramatic changes to its relative position and final EAII score.

5 Experiments

Since AlphaCode 2 is not publicly accessible, we chose to conduct our experiments using the available preview of OpenAI's new model, code-named o1-preview [1]. Unlike AlphaCode 2, which is proprietary to Google DeepMind, the o1-preview model was readily accessible via API calls. Consequently, in our experiments with ECN 2023 problems, we used o1-preview as the engine for the ECN AI 2023 (EAI2023) solver.

5.1 ECN 2023 edition

The 2023 edition of the Sapientia-ECN competition took place on November 20, bringing together 20 teams from Romania and Hungary, including 13 university teams and 7 high school teams. Participants faced a diverse set of 14 problems, ranging from relatively straightforward to highly challenging. The tasks spanned a variety of topics, with a particular focus on dynamic programming (Tasks A, F, I, N), classical algorithmic challenges (Tasks C, D, E, G, J), and problems involving mathematical concepts or data structures (Tasks B, H, K, L, M). Several problems required advanced techniques and optimizations, such as binary search (Tasks A,

[1] `https://openai.com/index/introducing-openai-o1-preview/`

M) and precomputations (Tasks E, M). The contest demanded not only strong algorithmic skills but also efficient implementations, posing a significant challenge even to the most experienced competitors.

5.2 ECN 2023 problems

Below we present the essence of the proposed tasks. For the exact problem statements visit the official website of the Sapientia-ECN programming competition [2].

5.2.1 Problem A: Gifts

Objective: Determine the optimal node in an induced subtree to add to a given list of 'generator nodes' to get the closest possible sum to a specified value.

- Tree Structure: An undirected tree with N vertices, where Fanurie visits each node in DFS order starting from node 1 and leaves gifts of increasing values.
- Subtree Induction: For each query containing a list of 'generator nodes', determine the minimal subtree that includes each node in the list (the subtree of the DFS tree rooted at the lowest common ancestor of the nodes in the list) and identify the node in the induced subtree that, when added, brings the list's total gift value closest to a given sum S.
- Constraints: Multiple queries with varying K (number of generator nodes in the list) and S (target sum), ensuring $K + 1$ nodes in the induced subtree.
- Output: For each query, the optimal node to add, ensuring it is not already in the list and choosing the smallest node in case of ties.

5.2.2 Problem B: Colors in Store

Objective: Determine the price each customer will pay for a tablecloth that matches their color preference or indicate if no matching tablecloth is available.

- Each tablecloth has a unique price and two color attributes (front and back) represented by integers from 1 to 3.
- Customers have a single favorite color and will purchase the cheapest available tablecloth with that color on either side.

[2]https://ecn.ms.sapientia.ro/problems.php

- Customers are served sequentially, and each purchase removes a tablecloth from the available options.
- For each customer, record the price paid or indicate if no suitable tablecloth is available.

5.2.3 Problem C: Golden Primes

Objective: Write a program to identify all values of n for which p, a golden prime, is less than a given threshold b.

- Golden Prime Definition: A golden prime is a prime number of the form $p = \phi^2 - \phi - 1$, where $\phi = 2^n$ and n is a positive integer.
- Input: A single positive integer b (where $b < 2^{1000}$), serving as the upper bound for p.
- Output Requirements: Print each value of n (one per line) for which p is a golden prime and less than b.

5.2.4 Problem D: Egyptian Fractions

Objective: Develop an algorithm to compute all possible Egyptian fraction representations for a given fraction, using the fewest possible terms.

- Egyptian Fraction Definition: An Egyptian fraction is a representation of a fraction as a sum of distinct unit fractions (fractions with numerator 1), such as $\frac{17}{70} = \frac{1}{5} + \frac{1}{24} + \frac{1}{840} = \frac{1}{7} + \frac{1}{10}$
- Input: Several lines, each containing a fraction in the form $\frac{a}{b}$, where a and b are positive integers and $0 < a < b$.
- Output Requirements: (i) Each line should display the minimum number of unit fractions in the Egyptian representation, followed by the denominators in lexicographically ordered parentheses; (ii) If multiple minimal representations exist, list each distinct set of denominators in lexicographical order.

5.2.5 Problem E: F. Lanovka – The cable car

Objective: Determine the minimum number of units needed to increase the heights of selected peaks to install a cable car system of length K with smoothly ascending columns in the Vector Vista Mountains.

- Peak Requirements: There are N peaks in total, and columns need to be installed on K consecutive peaks.
- Column Heights: The highest point of the cable car columns (peak + column) should reach height H, with subsequent heights on the remaining $K - 1$ columns descending by 1 unit each (i.e., $H - 1$, $H - 2$, ..., $H - K + 1$).
- Direction of Installation: The cable car path starts from the rightmost peak and goes leftward.

5.2.6 Problem F: Kings Again

Objective: Calculate the number of ways to place K kings on an $N \times M$ chessboard without them attacking each other.

- Chessboard Rules: Kings can attack each other if they are on adjacent cells (horizontally, vertically, or diagonally).
- Input: Multiple test cases, each specifying the values of N, M, and K.
- Constraints: Each test case requires calculating the number of valid placements modulo $10^9 + 7$.
- Output: For each test case, the number of ways to place the K kings.

5.2.7 Problem G: Breaking a Quantum Cryptography Machine

Objective: Analyze intercepted cryptographic messages and their solutions to deduce the operating principles of an enemy quantum cryptography machine and reimplement its functionality in C/C++.

- Intercepted Messages: Access to captured messages along with their verified solutions, provided by the Secret Service.
- Objective of Analysis: Identify and understand the underlying encryption principles used by the quantum cryptography machine based on the given examples.

5.2.8 Problem H: ”Optimizing” Ascent: Navigating Bear-Dangerous Territory in a Binary Matrix Mountain

Objective: Calculate the minimum number of steps taken through dangerous (internal) regions to ascend from the base to the summit of a multi-level ”mountain” represented by concentric binary matrices.

- Mountain Representation: The mountain has n levels, each encoded as an $m \times m$ binary matrix where: (i) A value of 1 represents mountain points; (ii) Each level's 1s form a connected region, including boundary points.
- Concentric Structure: Levels are "concentric", meaning if a point is 1 at level i, it is also 1 at level $i-1$, for $i = 2, \ldots, n$.
- Summit Configuration: The top level (last matrix) has a single 1, marking the mountain peak.
- Climbing Constraints: (i) Start at any boundary (edge) point on level 1; (ii) Move from the edge of each level to the next level's edge, aiming to reach the summit in the fewest steps through internal points; (iii) Moving between levels or along any edge is considered safe, as bears avoid these regions.

5.2.9 Problem I: Hamilton

Objective: Calculate the shortest route Hamilton, Santa's favorite elf, must take to deliver packages to all the mailboxes on a specific stretch of road.

- Coordinates: The road is 10000 meters long and 15 meters wide. Each mailbox has known coordinates (x, y) where $0 \leq x \leq 10000$ and $0 \leq y \leq 15$. The x-coordinates (abscissas) are unique for each mailbox, but the y-coordinates (ordinates) may repeat.
- Movement Constraint: Hamilton must traverse the mailboxes such that their x-coordinates first continuously increase and then continuously decrease, ensuring an efficient round trip.
- Start and End: The route starts at the first mailbox, covers all mailboxes, and returns to the starting point.
- Output: A single real number representing the length of the shortest route, printed with exactly six decimal places.

5.2.10 Problem J: Find the Identical Twins and Triplets

Objective: Identify and display groups of identical twins or triplets from a population database ($n \leq 160000$) who share the same DNA signature, including only those groups where at least one member is adopted.

- Database Structure for each individual: Personal Code (a unique 31-bit positive integer); DNA Signature (a string of up to 11 uppercase letters); Adoption Status (character 'A' for adopted, '$-$' otherwise); Name (up to 27 characters, may include whitespace).
- Unique Code Assignment: Consecutive personal codes in ascending order may have gaps, especially after assigning codes to twins or triplets. (Twins and triplets with the same DNA signature are guaranteed to appear consecutively in the sorted database by personal code)
- Grouping Requirement: (i) Identify groups of individuals with the same DNA (identical twins or triplets); (ii) Display only those groups containing at least one adopted member.

5.2.11 Problem K: Dependencies

Objective: Develop a program to determine if specified sequences of AWS resources, as defined in a CloudFormation template with dependencies, can be created in the given order. Identify any problematic resources that cannot be created due to dependency constraints.

- The input includes
  - R $(1 \leq R \leq 100)$ lines with resources and their dependencies in the format: Resource ID (alphanumeric, up to 20 characters) followed by a colon (":"), and then a space-separated list of its dependencies;
  - S $(1 \leq S \leq 100)$ lines with space-separated resource IDs representing the order in which resources are intended to be created.
- Output Requirements:
  - If all resources can be created in order, print "No problematic resources".
  - If some resources cannot be created in order: (i) For one problematic resource, print "Problematic resource: " followed by the resource ID; (ii) For multiple problematic resources, print "Problematic resources: " followed by the space-separated list of all problematic resource IDs in the order they appear in the sequence.

5.2.12 Problem L: Windmill-lottery

Objective: To simulate a national lottery system in ”Windmillia” and output the result of each draw based on specified rules. The simulation involves generating numbers and performing draws as per certain calculations, with numbers being assigned to two lines ($Line_1$ and $Line_2$) or used in a draw depending on modular conditions.

- Line and Number Generation:
  - Numbers are generated based on provided formulas involving constants a, b, c, and d.
  - Line assignment is determined by another formula with constants L_a, L_b, L_c, and L_d.
  - If $Line_n$ value is 1 or 2, $Number_n$ is assigned to the corresponding line. If $Line_n$ is 3, a draw is performed if the two lines have the same parity of count.
- Draw Process:
  - Condition for Draw: A draw can only occur if both $Line_1$ and $Line_2$ contain an even or odd number of elements.
  - Sorting: Both lines are sorted before the draw.
  - Rotation: $Line_2$ is rotated by rotationCount based on the calculated middle point.
  - Offset Calculation: An offset $Offset_i$ is calculated to find specific positions from the middle point in each line.
  - Winning Number: Using the middle point and offset, two values are summed (taking 0 if a line is too short), and their modulo m is taken as the winning number.
- Special Calculations:
  - Middle Point Determination: Adjustments are made for odd/even counts in each line.
  - Modulo m: Where m is the maximum value currently present across both lines.

5.2.13 Problem M: Modpute

Objective: To simulate the Modputer machine's behavior and calculate the total number of updates applied to its internal state array after processing a sequence of input integers. Each update occurs when an element in the state array is divisible by the current input integer.

- Internal State Array (A):
  - The machine's internal state is represented by an array A of N positive integers.
  - Each element in this array is subject to updates based on divisibility checks against the input integers.
- Input List:
  - A sequence of M positive integers, each greater than 1, is processed one by one.
  - For each integer D in this input, the machine checks each element in A to see if it is divisible by D.
- Update Mechanism:
  - For each integer D in the input: If an element $A[i]$ in the state array is divisible by D, it is incremented by one.
  - The operation is repeated for each integer in the input list, potentially causing multiple updates to each element in A.
- Count of Updates: Track the number of updates for each element in A and calculate the total number of updates across the entire array.

5.2.14 Problem N: Carpathian Riders

Objective: Plan an optimal motorcycle trip across the Carpathian Mountains on a grid of elevations.

- Grid Structure: The grid has R rows and C columns, with elevation values between 0 and 1000, and impassable cells marked as -1.
- Movement: The gang can move east, northeast, or southeast, starting from any passable cell on the western edge and aiming to exit on the eastern edge.

- Constraints: The trip must visit exactly P mountain passes, where a mountain pass is a cell with a higher elevation than its eastern and western neighbors and a lower elevation than its northern and southern neighbors.
- Goal: Minimize the sum of elevations along the trip while meeting the pass constraints.
- Output: The minimum sum of elevations for a valid trip or "impossible" if no such trip exists.

6 Results and Discussion

Metric	Value
Students Maximum Problems Solved ($\mathrm{Max}(PT_{\text{students}})$)	8
Students Minimum Problems Solved ($\mathrm{Min}(PT_{\text{students}})$)	0
Students Mean Problems Solved (Student mean PT)	2.47
Students Median Problems Solved ($\mathrm{Median}(PT_{\text{students}})$)	2
Students PT Standard Deviation ($\mathrm{STD}(PT_{\text{students}})$)	2.34
EAI2023 Total Problems Solved (PT_{AI})	10
EAI2023 Solved Percentage (SP)	71.42%
EAI2023 Partially Solved	3
Relative Performance to Median (as defined in Equation 1)	400%
Relative Performance to Peak (as defined in Equation 2)	25%
2023 ECN AI Baseline Index (as defined in Equation 3)	**128.57%**

Table 1: Key metrics

Performance statistics and key metrics are presented in Table 1. The problem-solving rates for each student team and the outcomes of EAII2023 are shown in Table 2. Additionally, the distribution of the number of problems solved by student teams is illustrated in Figure 1.

The problem-specific solving rates are visualized in Figure 2, which indicates varying levels of difficulty across different problems. Problems A and B were solved exclusively by student teams, while Problems C and D were solved solely by the AI system.

The box plot in Figure 3 provides a statistical summary of student performance, and Figure 4 reveals the correlations between different problems.

The results of this study underscore distinct problem-solving patterns between student teams and the EAI2023 AI system, particularly in terms of problem-specific

Problem	EAI2023 Solved	Student Teams Solved	Percentage of Teams Solved
A	Partially	1	5.26%
B	Partially	6	31.57%
C	Yes	0	0%
D	Yes	0	0%
E	Yes	7	36.84%
F	Yes	1	5.26%
G	Yes	16	84.21%
H	Yes	2	10.52%
I	Yes	2	10.52%
J	Yes	3	15.78%
K	Yes	4	21.05%
L	No	0	0%
M	Partially	0	0%
N	Yes	5	26.31%

Table 2: Problem-solving rates

strategies and advanced techniques. Metrics such as Relative Performance to Median (RPM) and Relative Performance to Peak (RPP) allow us to quantitatively assess the strengths of EAI2023 relative to student performance. Furthermore, qualitative analysis offers additional insights into how the unique requirements of each task shaped the outcomes, revealing important pedagogical implications for training in competitive programming.

6.1 Quantitative Analysis of Performance

As shown in Table 1, EAI2023 demonstrated outstanding performance, outperforming the median student team by an impressive 400% (RPM, Equation 1) and excelling across nearly all tasks. With a record-breaking 10 solved tasks, it exceeded the peak performance of the top student team by 25% (RPP, Equation 2), further cementing its reputation as a highly capable and effective problem-solving system.

The problem-specific analysis in Figure 2 reveals significant variances in difficulty across problems. Problems such as C and D, which were solved exclusively by EAI2023, highlight areas where the AI may have distinct advantages in systematic problem-solving under strictly defined test conditions. Conversely, Problems A and B, which were partially solved by students but not fully by EAI2023, may indicate situations where students displayed creative or heuristic-driven problem-solving abilities that the AI did not replicate fully. Problem G stands out as the most

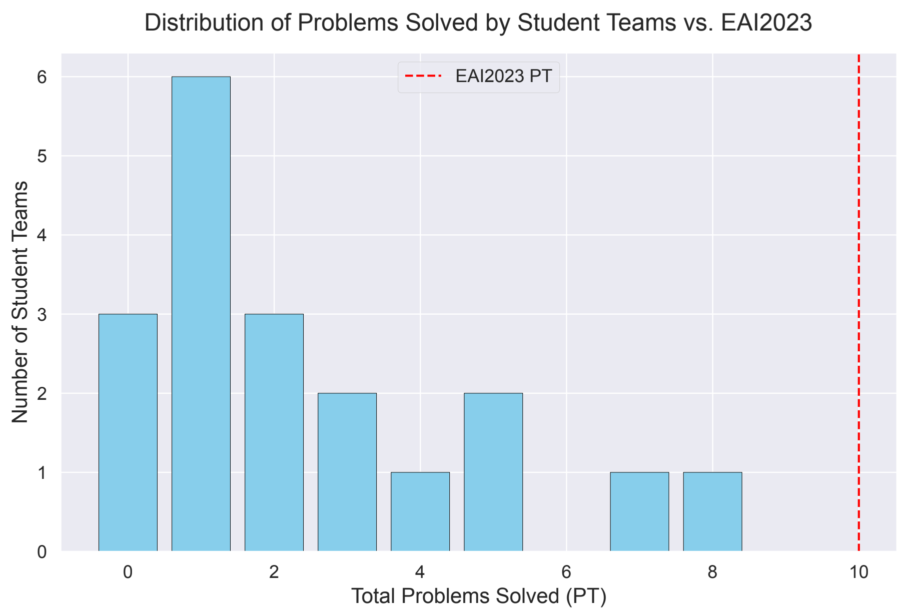


Figure 1: Distribution of total problems solved by student teams compared to EAI2023. The red dashed line indicates EAI2023's performance.

frequently solved problem among students (84.21% of teams), suggesting that certain types of problems align more closely with human problem-solving strategies than with AI-driven methods.

Figure 3 further illustrates the distribution of total problems solved by students, with the red dashed line marking the AI's score of 10 problems. The spread of student performance, reflected in a standard deviation of 2.34, points to a wide range of competencies among participants. This variability, combined with the solid performance of AI, suggests potential opportunities for future AI improvements aimed at bridging the gap in partially solved or creative problem types.

Finally, Figure 4 provides an overview of problem-solving correlations, though these patterns should be interpreted with caution due to the small sample size. For example, the observed perfect correlation between Problems H and I likely stems from these problems being solved only by the winning team, rather than indicating a significant general trend. Nevertheless, exploring such correlations in larger datasets could offer insights into how certain types of problems may cluster or interact, potentially guiding further AI development.

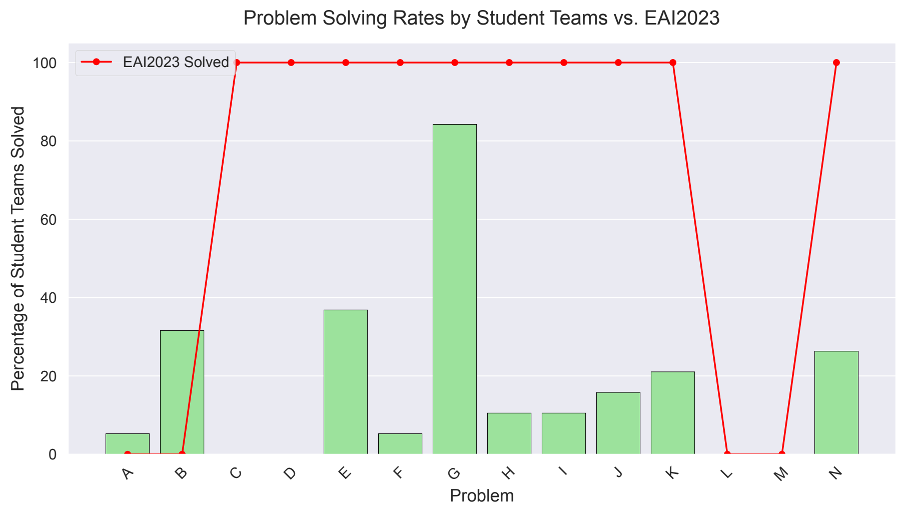


Figure 2: Problem-specific solving rates comparing student teams' performance with EAI2023. The green bars represent the percentage of student teams that solved each problem, while the red line shows whether the EAI2023 solved the problem or not.

6.2 Qualitative Analysis of Performance

The efficient solution of four tasks (A, F, I, N) required the application of a specific programming technique: dynamic programming (DP). Notably, the AI correctly identified the need for dynamic programming in all instances. For task N, the AI provided a valid solution in the first round, but it fell short in terms of speed for one test case, likely due to generating Python code, which is generally slower than languages like C. The AI correctly determined that a 3D DP array $dp[r][c][p]$ should be used, where r represents the row index, c the column index, and p the number of passes visited so far. In round 2, aiming to optimize performance, the AI tried to enhance the solution by implementing Dijkstra's algorithm. However, the Python version of this approach was even slower, resulting in a "time limit exceeded" error for two test cases. Learning from this, the AI returned to the dynamic programming approach and chose a more efficient data structure in Python, which ultimately resulted in a solution that was accepted for all test cases. Among the teams, five successfully solved this problem, with one team submitting a correct solution on their first attempt.

Task I was a specific variation of the Traveling Salesman Problem (TSP), known as the Bitonic TSP. Although this was not explicitly mentioned in the problem de-

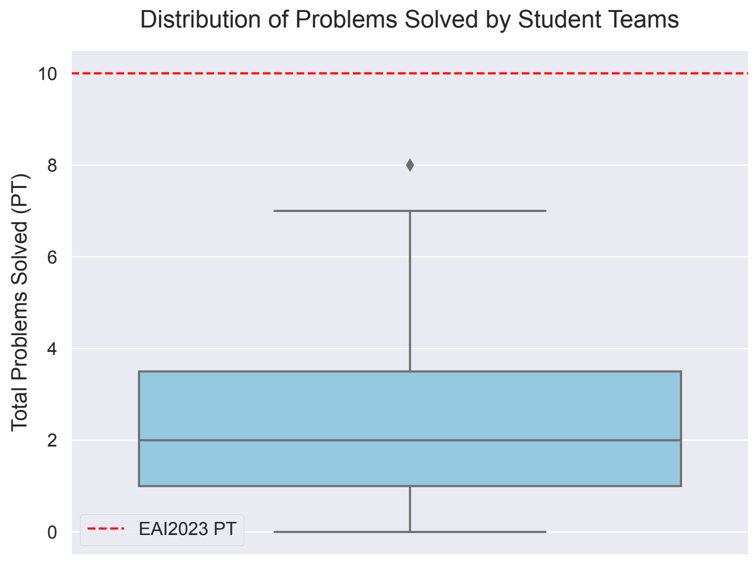
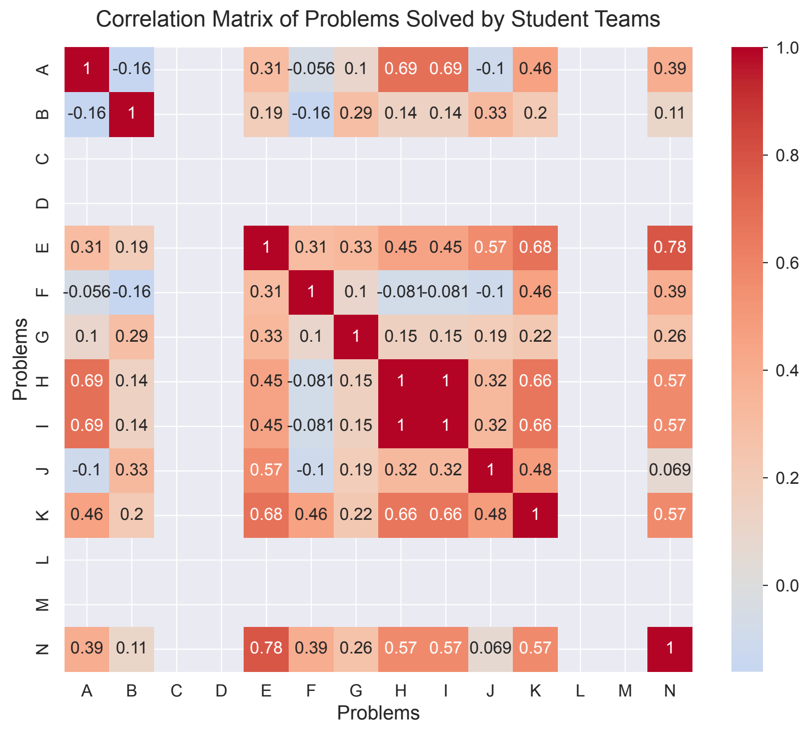


Figure 3: Box plot showing the distribution of total problems solved by student teams. The red dashed line indicates EAI2023's score, with 10 problems solved.

scription, the AI quickly recognized the nature of the problem. In the first round, it correctly formulated the recursive equations for computing the Bitonic Hamiltonian Paths but made an error when extracting the optimal length of the Bitonic Hamiltonian Cycle from the completed DP table. The AI recognized that it was dealing with a two-dimensional DP problem and accurately identified the general structure of the bitonic path subproblems: $dp[i][j]$ represented "the minimal total distance of a path that starts at mailbox 0 (leftmost), goes to mailbox i, then to mailbox j, visiting all mailboxes from 0 to j exactly once". When populating the DP array, the AI correctly distinguished between two cases: when i and j are consecutive mailboxes in the sorted order, and when they are not. In the second round, it successfully fixed the bug by ensuring that, when calculating the minimal total distance (the length of the Bitonic Hamiltonian Cycle), it considered all possible bitonic Hamiltonian paths ending at the rightmost point, not just those starting from the leftmost point. Only two teams managed to solve this problem—the first and second place teams—both on their first attempt.

Task F was also straightforward for the AI to recognize, as it involved a classic chessboard problem: determining the number of ways to place k kings on an $n \times m$ chessboard without them attacking each other. In the first round, the AI employed a similar approach to the one proposed by the task's author, who sug-



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	1	-0.16			0.31	-0.056	0.1	0.69	0.69	-0.1	0.46			0.39
B	-0.16	1			0.19	-0.16	0.29	0.14	0.14	0.33	0.2			0.11
C														
D														
E	0.31	0.19			1	0.31	0.33	0.45	0.45	0.57	0.68			0.78
F	-0.056	-0.16			0.31	1	0.1	-0.081	-0.081	-0.1	0.46			0.39
G	0.1	0.29			0.33	0.1	1	0.15	0.15	0.19	0.22			0.26
H	0.69	0.14			0.45	-0.081	0.15	1	1	0.32	0.66			0.57
I	0.69	0.14			0.45	-0.081	0.15	1	1	0.32	0.66			0.57
J	-0.1	0.33			0.57	-0.1	0.19	0.32	0.32	1	0.48			0.069
K	0.46	0.2			0.68	0.46	0.22	0.66	0.66	0.48	1			0.57
L														
M														
N	0.39	0.11			0.78	0.39	0.26	0.57	0.57	0.069	0.57			1

Figure 4: Correlation matrix showing the relationships between different problems. Darker red colors indicate positive correlations, while darker blue colors indicate negative correlations. The values range from -1 (perfect negative correlation) to 1 (perfect positive correlation).

gested an $O(n^2m^22^{2m})$ solution. However, this approach alone was not sufficient to meet the time limit. The author applied a clever trick, embedding precomputed results into the code. After we informed the AI of this optimization, it successfully implemented the trick. Interestingly, one of the competing teams devised a faster solution ($O(nmkC^2 + tC)$), t: number of test cases) that did not rely on the precomputed trick. Moreover, after the competition, a member of the competition committee further optimized this solution.

For task A, efficiently solving one of the subtasks required the use of dynamic programming, specifically through the technique of binary lifting—a method that was not widely known among the competitors. (Only the team trained by the author successfully solved this task.) Despite this, the AI immediately recognized the need for binary lifting to calculate the Lowest Common Ancestor (LCA) and applied it correctly. However, it misunderstood the task's definition of the "subtree induced by a list of nodes", leading to an incorrect solution in the first round. In the second round, while the AI identified its earlier mistake, it failed to recognize that this correction opened up an opportunity for further optimization. Instead, it modified its previous approach to fit the new understanding, resulting in an algorithm that still wasn't fast enough. The AI missed the fact that the nodes of the induced subtree form a consecutive segment in the DFS-ordered node list, and as a result, the corresponding value list was sorted in increasing order, making it possible to apply a binary search algorithm.

Two tasks required a solid mathematical background. Task C involved a seemingly simple prime number test but for extremely large numbers. The author provided a Python solution for this task, a language not permitted in the competition, and relied on the $isprime$ function, which is known to use a probabilistic approach beyond a certain threshold. This made the task's inclusion in the competition somewhat controversial. In the first round, the AI also used the $isprime$ function. However, since the competition's official judging system, although allowing Python submissions outside the competition, did not support the $sympy$ library (which $isprime$ depends on), we informed the AI of this limitation. In the second round, the AI explicitly implemented the Miller-Rabin test with 20 iterations for reliability, producing code that returned an "accepted" result for all test cases. Unsurprisingly, none of the teams solved this task, and understandably, only those with limited competition experience attempted it.

For task D, the AI quickly recognized the need to dynamically generate a rooted tree and search for the shortest path from the root to a solution leaf. The author's solution used a BFS search since the goal was to find the shortest decomposition. The main challenge was limiting the number of branches at each node. The author applied a clever trick, based on mathematical results vaguely hinted at in the problem description: estimating the maximum depth of the tree with a relatively small value and using this estimate to determine the maximum number of branches each node could have. Interestingly, the AI also realized that the solution depended on determining the maximum possible denominator for the next term based on the estimated tree depth. However, instead of guessing this limit, the AI repeatedly generated the tree and performed searches at increasing depths —1, 2, and so on— until it found a solution. In this approach, it was natural for the AI to initiate a DFS search in each iteration. The AI's code passed all test cases on the first attempt. During the competition, only one team attempted this task, but they were unsuccessful.

Many tasks required the use of appropriate built-in data structures, combined with efficient access algorithms like binary search, to achieve correct, efficient, and elegant solutions. These tasks were particularly well-suited to experienced competitors but also aligned with the AI's strengths. For instance, task B naturally called for a set, task E for an interval or segment tree, task G for a stack, task L for a multiset, and task M for a vector of sets, among others. Another common feature across several tasks was that certain precomputations significantly accelerated the algorithms. Examples include task E (precomputing sums and maximums for all consecutive fixed-length segments), task I (precomputing and storing all Euclidean distances), and task M (precomputing and storing all divisors to avoid repeated divisibility checks). These tasks favored both seasoned competitors and the AI alike.

Task G was the most frequently solved problem, with 16 out of 19 teams completing it. The challenge was to recognize that the problem revolved around the stack data structure, though it was presented in a coded form based on the given examples. Teams had to implement this solution, either by using the built-in stack structure or by directly coding it, for instance, with an array. Unsurprisingly, the AI successfully solved this task on its first attempt.

Task K was fairly straightforward, as it required implementing the algorithm directly from the problem description. No advanced optimizations were necessary due to the relatively small input size. Three teams—all of them podium finishers—worked on this task, with each solving it correctly. Both the first- and second-place teams succeeded on their first attempt, as did the AI. It is likely that the highly technical nature of the problem statement discouraged other teams from attempting it, as it appears they did not even engage with the task (at least, no solutions were submitted).

Another task that the AI solved on its first attempt was Task H. The author's proposed solution used a modified version of the Lee algorithm, which is generally well-known among experienced competitors. Due to the relatively small input size, no additional optimizations were needed to meet the time limit. The AI similarly employed a BFS-based approach, which is the core of the Lee algorithm. The top two teams also solved this task successfully on their first submissions. Meanwhile, one other team attempted it, submitting their code 12 times unsuccessfully—making it the most attempts for any task in the competition.

Task J presented a relatively simple challenge. The key requirement was efficient sorting, and once the array was sorted, the solution could be easily extracted, as the relevant items would naturally align next to each other. Despite 11 teams attempting this problem, only 3 succeeded in scoring points. It was later revealed that there was an error in the input file, and depending on the chosen strategy, teams either encountered or bypassed this issue. In the first round, the AI overlooked a crucial detail in the input specification, leading to faulty code. Although it corrected this in the second round, it once again missed the opportunity to leverage the benefits of a proper understanding of the problem, as it had with Task A. Instead of optimizing its approach, it merely extended the previous method to the new situation.

Nonetheless, the AI's proposed solution, though unnecessarily complex, ultimately proved correct and fast enough.

Task B was attempted by nearly every team (17 in total), but only 6 managed to receive an "accepted" result. The author sorted the tablecloths into three sets, as their fronts or backs could only fall into one of these categories. Interestingly, the AI also identified this as a natural approach, but instead opted for a more complex Python data structure in the hopes of creating a more efficient algorithm. However, it became overly complicated and failed to fix the solution even in the second round.

Task E was solved by 7 teams, making it the second most-solved problem. The author used an interval tree to efficiently find the maximum in different segments. The AI, however, provided a sufficiently fast solution by cleverly utilizing the *deque* data structure. While it made an error during the first round's maximum calculation, it corrected the mistake in the second round.

Tasks L and M were not inherently difficult but rather complex in their formulation. The convoluted wording of Task L may have discouraged participants, as no submissions were made for this problem. The author's solution involved implementing a custom heap structure, though the competition committee later introduced simpler alternatives, one of which relied on the built-in *multiset* data structure. For full context, it's worth noting that the creators of these alternative solutions had to consult the original task author due to misunderstandings or incorrect interpretations of the task description. This ambiguity may explain why the AI-generated code failed to produce output and likely contributed to teams' hesitation to engage deeply with the problem, resulting in no submissions.

For Task M, a straightforward naive solution was relatively easy to implement, but the challenge lay in optimizing each subtask and applying various advanced implementation techniques. One such technique involved precomputing divisors, while another clever approach by the task author involved storing the positions or indices of input array elements in sets to enable efficient retrieval through binary search. The AI's initial solution produced incorrect results, even for the example provided in the problem description. In the second attempt, the AI corrected this issue, but its algorithm was still too slow for 3 out of 9 test cases. It seems that the AI's optimization strategies—such as associating frequency arrays with both input arrays and using the Sieve of Eratosthenes to precompute the smallest prime factor for all numbers up to MAX (300,000 in this case) to accelerate factorization—were insufficient in terms of both efficiency and precision. Only one prize-winning team managed to submit a solution for this task, but it was ultimately unsuccessful. Other teams that attempted the task multiple times, also without success, generally ranked lower on the leaderboard.

6.3 Pedagogical Insights and Lessons Learned

The analysis of AI solutions, task author approaches, and team performances reveals important pedagogical insights, especially valuable for developing effective competitive programming training. These insights can help shape training strategies and highlight common pitfalls to avoid.

6.3.1 Technique Familiarity vs. Flexibility in Approach

Certain tasks required advanced techniques like dynamic programming for tasks A, F, I, and N, and binary lifting specifically for task A. While both the AI and the task author correctly applied binary lifting for task A, only the author-trained team among competitors used it, underscoring the need for explicit training in specialized algorithmic strategies. Familiarity with less commonly known methods like binary lifting, which can bring considerable efficiency gains, is essential for competitors aiming to improve performance.

6.3.2 Programming Language Constraints and Optimization Awareness

Task C, which required large prime number tests, highlighted the impact of programming language limitations. Both the task author and AI initially used Python's *isprime* function from the *sympy* library, which was unavailable in the competition's environment. The AI's pivot to a custom Miller-Rabin test implementation emphasizes the importance of preparing students to handle unexpected constraints and limitations. Competitors should practice adjusting strategies based on language-specific features and know when to implement fallback methods.

6.3.3 Strategic Use of Precomputation and Data Structures

Tasks E, I, and M demonstrated the substantial advantages of precomputing values to optimize complex calculations. While the AI often identified opportunities for precomputation, the majority of teams did not, highlighting the need for training in recognizing precomputation opportunities early in problem-solving.

Additionally, task-specific data structures played a vital role in solution optimization (e.g., *sets* for task B, *deques* for task E). While the AI generally applied these structures effectively, many teams did not. For students, mastering data structures and quickly assessing their applicability in different scenarios can significantly enhance performance and reduce complexity.

6.3.4 Error Recovery and Revisiting Problem Understanding

In tasks A and J, the AI initially misunderstood problem requirements. Although it corrected these errors, it continued refining its original flawed approach instead of reassessing the solution with the clarified requirements. This tendency, common among learners, to stick to an initial solution path can hinder progress even after gaining new insights. Training should encourage students to consider a ”reset” approach once key misunderstandings are resolved, promoting a fresh perspective that could reveal simpler or more efficient solutions.

6.3.5 Clarity and Ambiguity in Problem Statements

Tasks L and M demonstrated how complex or ambiguous problem wording can be a significant barrier. Both competitors and the AI struggled with interpretation, especially for tasks requiring custom structures or specific mathematical insights. This underscores the importance of training competitors to deconstruct complex descriptions, focus on core requirements, and approach ambiguous problems with resilience, enabling them to tackle even the most challenging wording effectively.

6.3.6 Reinforcing Mathematical and Theoretical Background

Some tasks required specialized mathematical insights, such as understanding the bitonic Traveling Salesman Problem for task I or efficiently testing large prime numbers in task C. Few competitors attempted these problems, which highlights the value of strengthening students’ mathematical reasoning and theoretical knowledge as part of competitive programming training. Advanced-level training should incorporate more complex mathematical challenges to develop these critical skills among competitors.

6.4 EAII Advatnages and Limitation

EAII offers several advantages for longitudinal analysis. By normalizing performances, the EAII ensures comparability over time, remaining consistent across contests with differing problem sets and participant pools. This consistency enables meaningful longitudinal comparisons of AI capability advancements. Using the median student performance enhances robustness to outliers, reducing the influence of extreme values and skewed distributions and providing a more stable reference point, especially in datasets with small sample sizes. The metric also provides a compact insight, capturing the AI’s relative standing within the spectrum of student performance and offering insights into both average (median) and peak human capabilities. EAII's percentage scale enhances interpretability, facilitating straightforward communication of results.

EAII has certain limitations. It is dependent on the student performance distribution and sensitive to factors unrelated to the AI's capabilities, such as changes in participant demographics or preparation levels. In contests with a limited number of student teams, the median and peak may not accurately represent typical or maximum human performance, potentially skewing the EAII due to the impact of small sample sizes. The metric assumes a linear relationship between the median and peak performances, which may not capture the nuances of performance distributions, particularly if the spread is uneven or significant gaps exist between top performers and others. Additionally, the EAII focuses on two specific points—the median and peak—and does not account for the full distribution of student performances, which might provide additional context.

The EAII is inherently influenced by the total number of problems presented in a contest, as the sheer volume can affect the performance outcomes for both AI and human teams. In instances where a large number of problems are available, human teams may struggle to address all of them within the five-hour time limit of the contest. This limitation could inadvertently advantage the AI.

To address some of these limitations, incorporating additional statistical measures such as the interquartile range (IQR) might add more stability to the metric. The IQR considers the spread of the middle 50% of the data, making it less sensitive to extreme values. However, reliably computing quartiles requires an even larger sample size with many participating teams. In contests with a small number of participants, quartile calculations may be less robust, potentially limiting the effectiveness of IQR-based adjustments.

7 Conclusions

The ECN AI Performance Index (EAII) introduced in this paper provides a normalized, interpretable metric for comparing AI and human performances in competitive programming contests. By incorporating both median and peak performances, the EAII benchmarks the AI's capabilities against both average and top-performing competitors, offering a well-rounded perspective on AI's standing within human standards of excellence. This dual reference point makes EAII particularly valuable in identifying not just raw AI performance but also the areas where AI approaches human problem-solving strengths, as well as areas where it lags behind.

In this analysis, the EAII has proven effective in capturing the AI's competencies and limitations across a range of tasks, revealing both the current status and developmental potential of AI systems in this domain. For example, our qualitative analysis highlights that the AI is well-suited to problems requiring systematic approaches, such as dynamic programming or binary lifting. However, tasks involving flexible heuristics, creative partial solutions, or complex mathematical reasoning sometimes challenge the AI, indicating opportunities for further enhancement in these areas. By tracking such nuances over time, the EAII allows us to observe shifts in AI's problem-solving capabilities and to identify specific areas for potential improvement.

The EAII's design also accommodates variations in contest conditions and participant performance distributions, making it a robust tool for longitudinal analysis across multiple competitions. Although limitations exist—particularly with respect to sample size and distribution assumptions—EAII nonetheless provides an invaluable metric for evaluating the evolution of AI in competitive programming, tracking AI's progress with accessible metrics that reflect achievable benchmarks rather than optimal outcomes. This ensures that the EAII remains widely applicable and interpretable over time, as it is not tied to specialized optimization techniques that could obscure true progress in baseline AI capabilities.

Importantly, EAII is intentionally designed to reflect a consistent, achievable benchmark rather than the maximum potential performance attainable with highly advanced techniques. While agent-based systems [13]–[15], retrieval-augmented generation [16]–[18], or other sophisticated methods could indeed push AI capabilities further, our focus is on establishing a baseline that is accessible to anyone using straightforward prompts and publicly available models. This baseline provides a reference point for tracking AI progress over time without the confounding effects of specialized techniques and customized optimization strategies. In this way, the EAII supports transparent and meaningful assessment of AI progress, helping researchers and practitioners understand the real-world applicability and constraints of AI within competitive programming contexts.

Overall, EAII is a stepping stone for understanding AI's evolving role in problem-solving, offering insights that extend beyond mere performance scores. By capturing a nuanced view of AI's strengths, limitations, and growth areas, the EAII fosters a richer understanding of AI's development trajectory and its potential for reaching, and perhaps one day surpassing, human performance benchmarks in complex problem-solving domains.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflicts of interest.

Disclaimer: LLMs were used to enhance the phrasing and flow of this manuscript. However, all core content - including the research concept, experimental design, data analysis, results, and scientific interpretations - are solely the product of authors' work. The AI language models were used exclusively to improve the clarity and presentation of our research findings.

References

[1] Z. Manna and R. J. Waldinger, "Toward automatic program synthesis," *Communications of the ACM*, vol. 14, no. 3, pp. 151–165, 1971 (⇒ 257).

[2] S. Gulwani, O. Polozov, R. Singh, *et al.*, "Program synthesis," *Foundations and Trends® in Programming Languages*, vol. 4, no. 1-2, pp. 1–119, 2017 (⇒ 257).

[3] C. Green, "Application of theorem proving to problem solving," in *Readings in Artificial Intelligence*, Elsevier, 1981, pp. 202–222 (⇒ 257).

[4] Y. Li, D. Choi, J. Chung, *et al.*, "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022 (⇒ 257, 258).

[5] Q. Shi, M. Tang, K. Narasimhan, and S. Yao, "Can language models solve olympiad programming?" Tech. Rep., 2024, arXiv preprint arXiv:2404.10952 (⇒ 257).

[6] Y. Huang, Z. Lin, X. Liu, *et al.*, "Competition-level problems are effective llm evaluators," Tech. Rep., 2023, arXiv preprint arXiv:2312.02143 (⇒ 257).

[7] T. Coignion, C. Quinton, and R. Rouvoy, "A performance study of llm-generated code on leetcode," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, 2024, pp. 79–89 (⇒ 257, 258).

[8] B. Idrisov and T. Schlippe, "Program code generation with generative ais," *Algorithms*, vol. 17, no. 2, p. 62, 2024 (⇒ 258).

[9] AlphaCode Team, Google DeepMind, "AlphaCode 2 Technical Report," Google DeepMind, Tech. Rep., Dec. 2023, 7 pages (⇒ 258).

[10] The AlphaCode team. "Competitive programming with AlphaCode." Accessed: 2024-11-03. (Dec. 2022), [Online]. Available: https://deepmind.google/discover/blog/competitive-programming-with-alphacode/ (⇒ 258).

[11] OpenAI o1 Contributors. "CLearning to Reason with LLMs." Accessed: 2024-11-03. (Sep. 2024), [Online]. Available: https://openai.com/index/learning-to-reason-with-llms/ (⇒ 258).

[12] K. Valmeekam, K. Stechly, A. Gundawar, and S. Kambhampati, "Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of lrm o1," Tech. Rep., 2024, arXiv preprint arXiv:2410.02162 (⇒ 258).

[13] G. Sarch, Y. Wu, M. J. Tarr, and K. Fragkiadaki, "Open-ended instructable embodied agents with memory-augmented large language models," Tech. Rep., 2023, arXiv preprint arXiv:2310.15127 (⇒ 283).

[14] C. Li, H. Chen, M. Yan, *et al.*, "Modelscope-agent: Building your customizable agent system with open-source large language models," Tech. Rep., 2023, arXiv preprint arXiv:2309.00986 (⇒ 283).

[15] I. de Zarzà, J. de Curtò, G. Roig, P. Manzoni, and C. T. Calafate, "Emergent cooperation and strategy adaptation in multi-agent systems: An extended coevolutionary theory with llms," *Electronics*, vol. 12, no. 12, p. 2722, 2023 (⇒ 283).

[16] H.-r. Lee and S.-h. Kim, *Bring retrieval augmented generation to google gemini via external api: An evaluation with big-bench dataset*, Research Square, PREPRINT (Version 1), May 2024 (⇒ 283).

[17] W. Huang, M. Lapata, P. Vougiouklis, N. Papasarantopoulos, and J. Pan, "Retrieval augmented generation with rich answer encoding," in *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 1012–1025 (⇒ 283).

[18] Z. Shao, Y. Gong, Y. Shen, M. Huang, N. Duan, and W. Chen, "Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy," Tech. Rep., 2023, arXiv preprint arXiv:2305.15294 (⇒ 283).

Received: 05.11.2024; *Revised: 28.11.2024*; *Accepted: 28.11.2024*