SP25 CSE 5525 Final Project Presentation

AutoAC: Agentic Self-Refinement for Competitive Programming

Sriram Sai Ganesh, Robert Pham

What

Our *agentic framework* helps *researchers & engineers* who want to *use LLMs for code* by formalizing an approach to solving algorithmic programming problems.

AutoAC: Agentic Self-Refinement for Competitive Programming

LLM Agent-Computer Interface grounded in CP strategy

Team members

Ram Sai Ganesh

Robert Pham

Why

Competitive Programming (CP) problems are:

Challenging:
- Often require advanced computational thinking & problem-solving skills.

Objectively Gradable:
- Solutions can be quantitatively & automatically evaluated (hidden testcases)

Compositional:
- Large Language Models (LLMs) struggle with simple composite tasks*

Deliverables

AutoAC, a framework that supports:

✅ Prompting LLMs for CP tasks.

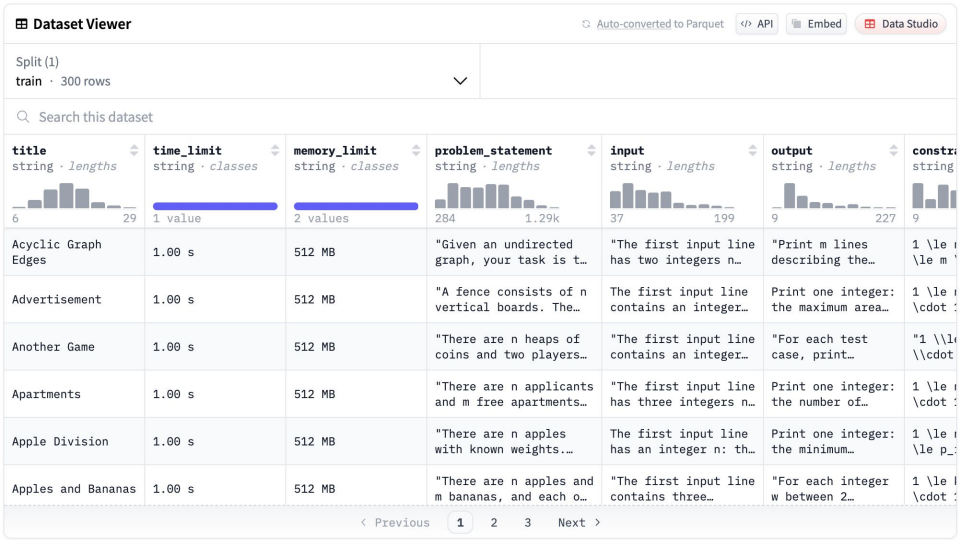
✅ Automatically evaluating LLM code.

✅ LLM-interpreter interface to self-debug.

⏳ Multi-agent cooperation for stress testing

How

juvi21 / **cses-fi-competitive-coding-problems**



Dataset Viewer

Auto-converted to Parquet | API | Embed | Data Studio

Split (1)
train · 300 rows

Search this dataset

title string · *lengths* 6 – 29	time_limit string · *classes* 1 value	memory_limit string · *classes* 2 values	problem_statement string · *lengths* 284 – 1.29k	input string · *lengths* 37 – 199	output string · *lengths* 9 – 227	constra string 9
Acyclic Graph Edges	1.00 s	512 MB	"Given an undirected graph, your task is t…	"The first input line has two integers n…	"Print m lines describing the…	1 \le r \le m '
Advertisement	1.00 s	512 MB	"A fence consists of n vertical boards. The…	The first input line contains an integer…	Print one integer: the maximum area…	1 \le r \cdot 1
Another Game	1.00 s	512 MB	"There are n heaps of coins and two players…	"The first input line contains an integer…	"For each test case, print…	"1 \\le \\cdot
Apartments	1.00 s	512 MB	"There are n applicants and m free apartments…	"The first input line has three integers n…	Print one integer: the number of…	1 \le r \cdot 1
Apple Division	1.00 s	512 MB	"There are n apples with known weights.…	The first input line has an integer n: th…	Print one integer: the minimum…	1 \le r \le p_1
Apples and Bananas	1.00 s	512 MB	"There are n apples and m bananas, and each o…	"The first input line contains three…	"For each integer w between 2…	1 \le \cdot 1

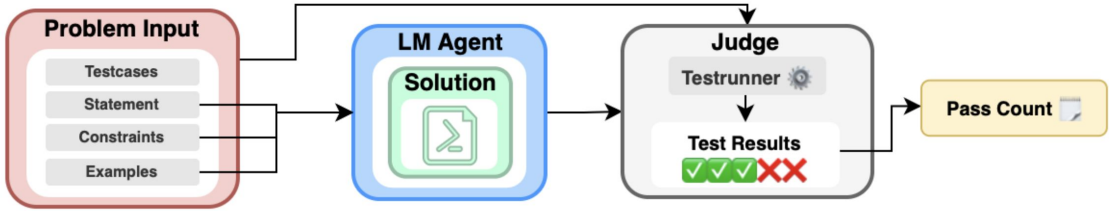
‹ Previous 1 2 3 Next ›

We tested on...

- Qwen2.5-0.5B-Instruct
- Llama-3.2-1B-Instruct
- Qwen2.5-1.5B-Instruct
- Llama-3.2-3B-Instruct
- Qwen2.5-3B-Instruct
- Qwen2.5-7B-Instruct
- CodeLlama-7b-Instruct-hf
- Llama-3.1-8B-Instruct
- CodeLlama-13b-Instruct-hf
- Qwen2.5-14B-Instruct

How

AutoAC



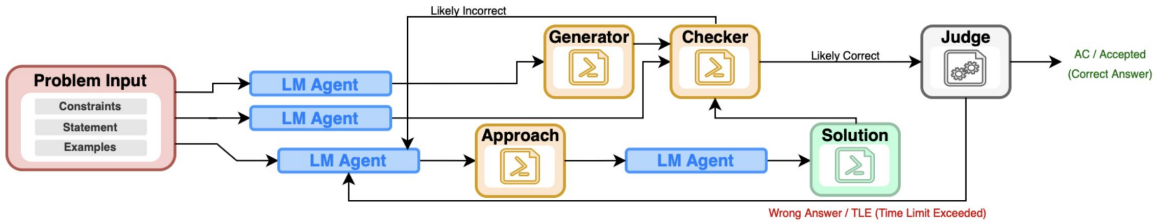
Phase one:

Measuring baselines.
How does a single prompt perform?

Phase two:

What happens when we add complex, agentic behavior?

- Automated retry attempts
- Automated self-testing of solutions
    - with self-debugging



Metrics

Solve Rate (SR)

AC – Problem Accepted (all tests passed)

N_{AC} – Total number solved

SR – Percent of problems solved

$$\mathrm{N_{AC}} \leftarrow \sum_{i=1}^{|D|} \mathbb{1}\left(\mathrm{AC}_i\right)$$

$$\mathrm{SR} \leftarrow 100 \cdot \frac{\mathrm{N_{AC}}}{|D|}$$

Normalized Testcase Pass Rate (NTPR):

$\mathrm{Partial}_{(i)}$ – per-problem proportion of passed testcases.

NTPR – Average proportion of testcases passed

$$\mathrm{NTPR} \leftarrow 100 \cdot \frac{\sum_{i=1}^{|D|} \mathrm{Partial}^{(i)}}{|D|}$$

Results

Raw Data (NTPR)

Large Language Model	Normalized Testcase Pass Rate (%)							
	A	B	C	D	E	F	G	H
Qwen2.5-0.5B-Instruct	2.01	1.62	1.64	0.87	5.09	3.94	5.85	**6.86**
Llama-3.2-1B-Instruct	3.85	7.48	7.65	7.97	6.77	7.20	7.69	**9.68**
Qwen2.5-1.5B-Instruct	4.48	5.61	4.08	6.37	9.49	5.47	**10.86**	10.05
Llama-3.2-3B-Instruct	9.98	11.35	13.52	13.32	13.23	13.36	16.39	**18.21**
Qwen2.5-3B-Instruct	14.70	12.88	12.65	17.15	17.56	14.32	16.93	**17.77**
Qwen2.5-7B-Instruct	23.11	25.04	24.56	25.85	**28.97**	23.85	26.49	28.29
CodeLlama-7b-Instruct-hf	12.05	12.92	6.90	7.82	12.67	7.64	10.72	**12.94**
Llama-3.1-8B-Instruct	20.19	17.83	16.78	17.26	20.12	14.82	19.64	**20.67**
CodeLlama-13b-Instruct-hf	15.55	16.64	14.87	11.31	12.98	10.55	**17.68**	17.36
Qwen2.5-14B-Instruct	28.80	26.06	28.48	**34.50**	29.30	29.57	28.81	32.60

Results

Raw Data (SR)

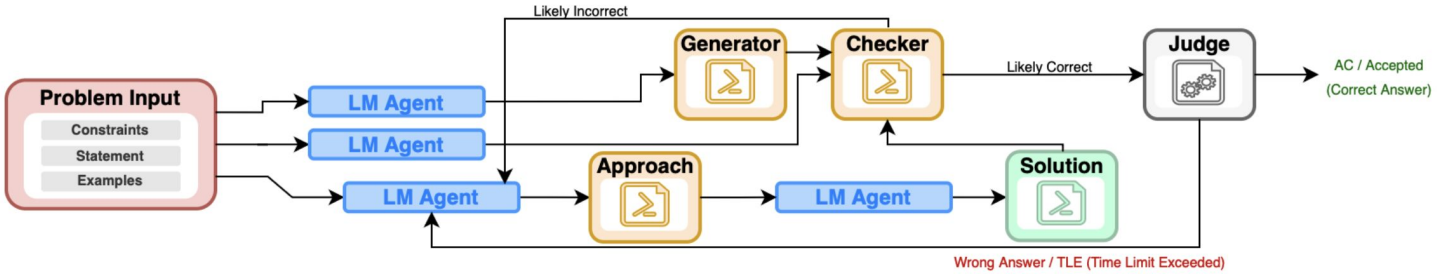
Large Language Model	Problem Solve Rate (%)							
	A	B	C	D	E	F	G	H
Qwen2.5-0.5B-Instruct	0.33	0.00	0.33	0.00	0.33	0.33	**1.00**	0.33
Llama-3.2-1B-Instruct	0.00	0.66	0.66	**1.00**	0.00	**1.00**	**1.00**	0.33
Qwen2.5-1.5B-Instruct	0.66	1.33	1.00	1.00	1.33	0.66	1.33	**1.66**
Llama-3.2-3B-Instruct	2.66	2.66	3.66	3.33	2.66	1.66	3.66	**4.33**
Qwen2.5-3B-Instruct	6.00	3.66	3.00	**7.33**	5.66	4.66	5.00	4.66
Qwen2.5-7B-Instruct	9.66	10.66	10.00	12.00	**13.66**	9.33	12.33	12.66
CodeLlama-7b-Instruct-hf	0.66	1.00	0.33	**1.66**	1.33	0.66	1.00	1.00
Llama-3.1-8B-Instruct	**7.00**	4.66	4.66	6.00	6.00	3.33	5.33	5.66
CodeLlama-13b-Instruct-hf	1.33	2.00	2.00	1.00	1.33	1.00	**3.00**	2.33
Qwen2.5-14B-Instruct	15.66	13.00	13.33	**18.66**	14.33	14.33	16.33	16.33

Trends Baseline vs. Best

Model Name	$NTPR_i$	$NTPR_f$	Δ	SR_i	SR_f	Δ
Q2.5-0.5B-I	2.01%	6.86%	(+4.85%)	0.33%	1.00%	(+0.67%)
Q2.5-1.5B-I	4.48%	10.86%	(+6.38%)	1.33%	1.66%	(+0.33%)
Q2.5-3B-I	14.70%	17.77%	(+3.07%)	3.66%	7.33%	(+3.67%)
Q2.5-7B-I	23.11%	28.97%	(+5.86%)	10.66%	13.66%	(+3.00%)
Q2.5-14B-I	28.80%	34.50%	(+5.70%)	13.0%	18.66%	(+5.66%)

Table 1: A NTPR & SR of baselines (i) vs most performant prompting approach (f).

Trends Stress Testing



Model Name	$NTPR_{NS}$	$NTPR_{S}$	Δ	SR_{NS}	SR_{S}	Δ
Q2.5-3B-I	17.77%	20.01%	(+2.24%)	7.33%	8.93%	(+1.60%)
Q2.5-14B-I	34.50%	37.84%	(+3.34%)	18.66%	19.57%	(+0.91%)

Table 2: NTPR & SR of most performant stress (S) vs non-stress (NS) approach.

Roadmap

TODOs

- Debug **AutoAC** stress tests.
- Analyze failure modes.
- Run stress-test experiments on full CSES dataset.

Future Work

- Expand eval datasets
- RL finetuning -- GRPO?
- Closed-source + larger models (budget limitations)

Thank you! Questions?

AutoAC: Agentic Self-Refinement for Competitive Programming

Sriram Sai Ganesh, Robert Pham

Experimental Groups

Experiment Tag	Experiment Label	Detailed Prompt Description
A	Baseline non-CoT	Instruct the model to solve the problem in C++
B	Baseline CoT	Instruct the model to reason through its solution.
C	Refined CoT	Step-by-step CoT Instructions
D	Domain-specific CoT	Detailed CoT including guidelines about CP best practices
E	ICL 1-shot OOD Short	ICL + CoT + 1 concise synthetic problem-solution pair
F	ICL 1-shot OOD Long	ICL + CoT + 1 complex synthetic problem-solution pair
G	ICL 1-shot ID Long	ICL + CoT + 1 in-domain (from training data) problem-solution pair
H	ICL 3-shot OOD Short	ICL + CoT + 3 examples of concise synthetic problem-solution pairs

Accepted!

Solve this CP problem titled “Advertisement”:

A fence consists of n vertical boards. The width of each board is 1 and their heights may vary. You want to attach a rectangular advertisement to the fence. What is the maximum area of such an advertisement?

Input

The first input line contains an integer n: the width of the fence. After this, there are n integers $k_1, k_2, \ldots, k_n$: the height of each board.

Output

Print one integer: the maximum area of an advertisement.

Constraints

$$1 \le n \le 2 \cdot 10^5$$
$$1 \le k_i \le 10^9$$

Example

Input:

```
8
4 1 5 3 3 2 4 1
```

Output:

```
10
```

- If the array has only one element, the maximum area is just that element's value since it cannot form a larger rectangle.

5. **Efficiency**: This approach runs in linear time, $O(n)$, making it suitable even for large inputs up to $2 * 10^5$.

By following these steps, we ensure our solution is both efficient and correct, handling the constraints and edge cases effectively.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int n;
    cin >> n;
    vector<int> heights(n);
    for(int i = 0; i < n; i++) cin >> heights[i];

    vector<int> left(n), right(n);
    stack<int> st;

    // Finding nearest smaller elements on the left
    for(int i = 0; i < n; i++) {
        while(!st.empty() && heights[st.top()] >= heights[i]) st.pop();
        if(st.empty()) left[i] = 0;
        else left[i] = st.top() + 1;
        st.push(i);
    }

    // Clearing stack for reuse
    while(!st.empty()) st.pop();

    // Finding nearest smaller elements on the right
    for(int i = n-1; i >= 0; i--) {
        while(!st.empty() && heights[st.top()] >= heights[i]) st.pop();
        if(st.empty()) right[i] = n-1;
        else right[i] = st.top() - 1;
        st.push(i);
    }

    long long maxArea = 0;
    for(int i = 0; i < n; i++) {
        long long width = right[i] - left[i] + 1;
        maxArea = max(maxArea, heights[i] * width);
    }

    cout << maxArea << endl;
}
```

Explanation: The solution employs a stack-based method to efficiently determine the maximum area of a rectangle that can be formed from the fence boards. It calculates the nearest smaller elements on both the left and right for each board, allowing it to compute the maximum width each board can span without violating the height constraint. By iterating through each board and calculating its potential contribution to the maximum area, the algorithm ensures optimal performance within the given constraints.

Wrong Answer.

Solve this CP problem titled “Another Game”:

There are n heaps of coins and two players who move alternately. On each move, a player selects some of the nonempty heaps and removes one coin from each heap. The player who removes the last coin wins the game. Your task is to find out who wins if both players play optimally.

Input

The first input line contains an integer t: the number of tests. After this, t test cases are described: The first line contains an integer n: the number of heaps. The next line has n integers $x_1, x_2, \ldots, x_n$: the number of coins in each heap.

Output

For each test case, print “first” if the first player wins the game and “second” if the second player wins the game.

Constraints

$$1 \le t \le 2 \cdot 10^5$$
$$1 \le n \le 2 \cdot 10^5$$
$$1 \le x_i \le 10^9$$

the sum of all n is at most $2 \cdot 10^5$

Example

Input:

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

Output:

```
first
second
first
```

Implementation:

Here's the detailed implementation:

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int t;
    cin >> t;

    while (t--) {
        int n;
        cin >> n;
        vector<int> coins(n);

        // Read the number of coins in each heap
        for (int i = 0; i < n; ++i) {
            cin >> coins[i];
        }

        // Check if all heaps have the same number of coins
        bool all_same = true;
        for (int i = 1; i < n && all_same; ++i) {
            if (coins[i]!= coins[0]) {
                all_same = false;
            }
        }

        // Determine the winner
        if (all_same) {
            cout << "second\n";
        } else {
            cout << "first\n";
        }
    }

    return 0;
}
```

This code efficiently reads the input, checks if all heaps have the same number of coins, and prints the appropriate winner for each test case. The solution leverages the observation that the game state determines the winner based on whether the heaps are uniform or not.