Can AI Judges Replace Unit Tests for Code Evaluation?

Evaluating LLM-as-a-Judge and Agent-as-a-Judge on Coding Tasks

By Lin Shi, Yan Xiao, Tongjia Rao

Cornell Tech, New York, NY, USA

Research Question & Motivation

Research Question

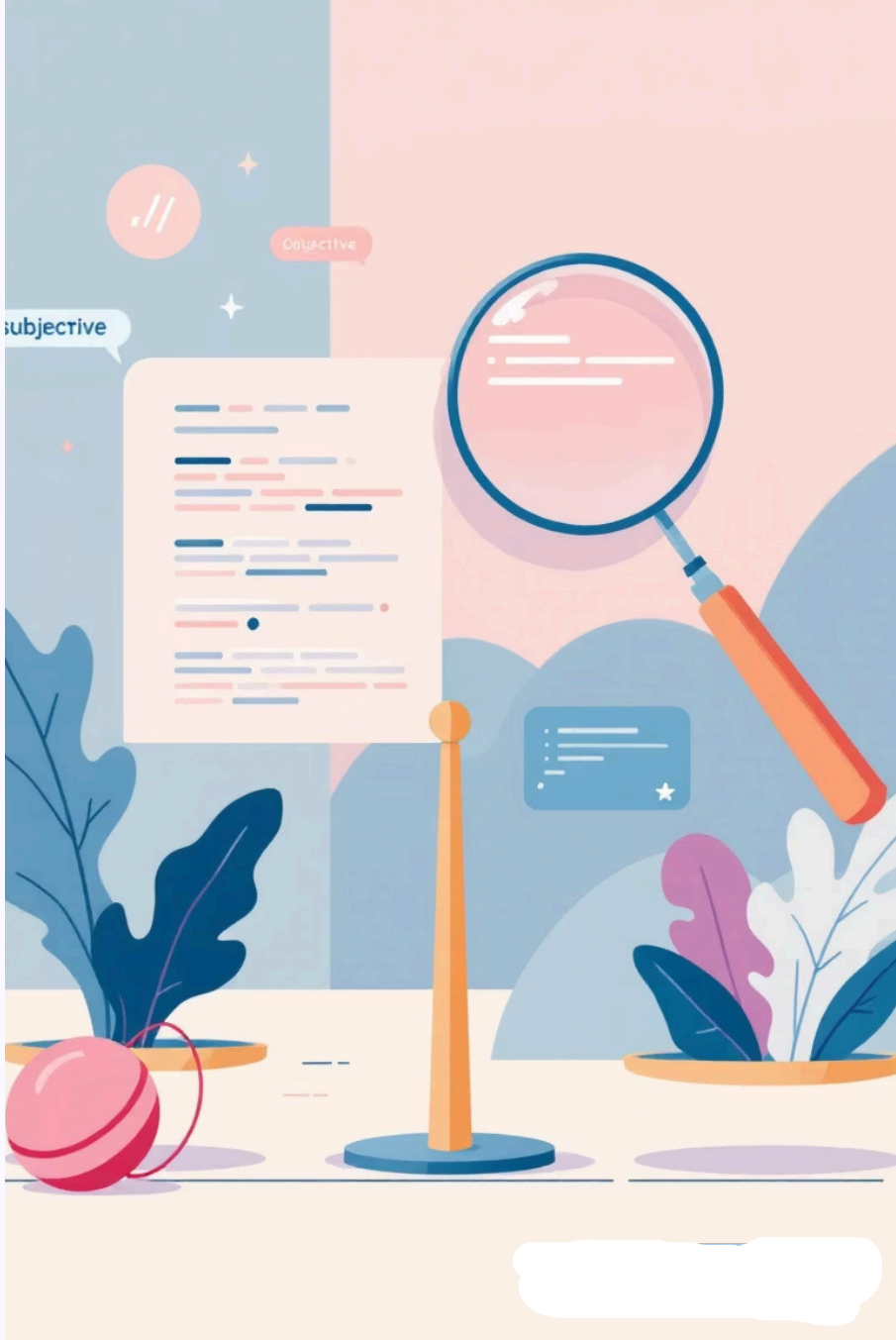
Can subjective AI evaluators (LLM-as-a-Judge, Agent-as-a-Judge) replace objective unit testing for code correctness evaluation?

Motivation

As LLM code generation scales, verifying correctness is critical but unit tests are labor-intensive and costly to maintain.

Why Important

Need scalable, cost-effective evaluation methods that preserve correctness signals while potentially offering richer insights.



Key Research Questions

1 **Signal Preservation**

Can LLM and agent judges faithfully preserve the correctness signal of human-authored unit tests?

2 **Subjectivity as Advantage**

Does their additional subjectivity provide richer evaluation without sacrificing reliability?

Data Collection

1 **Dataset**

70 LeetCode-style programming tasks from LiveCodeBench.

2 **Task Components**

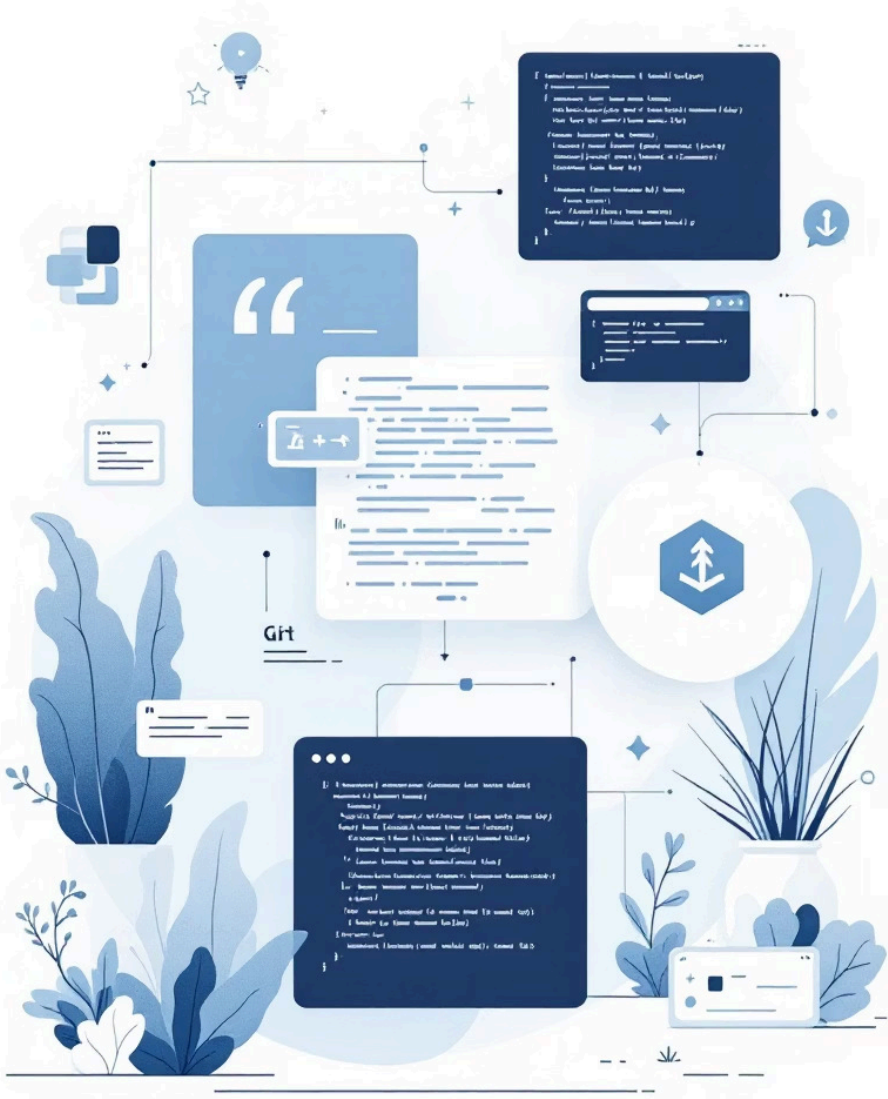
Each task includes: problem description, reference solutions, human-annotated unit tests.

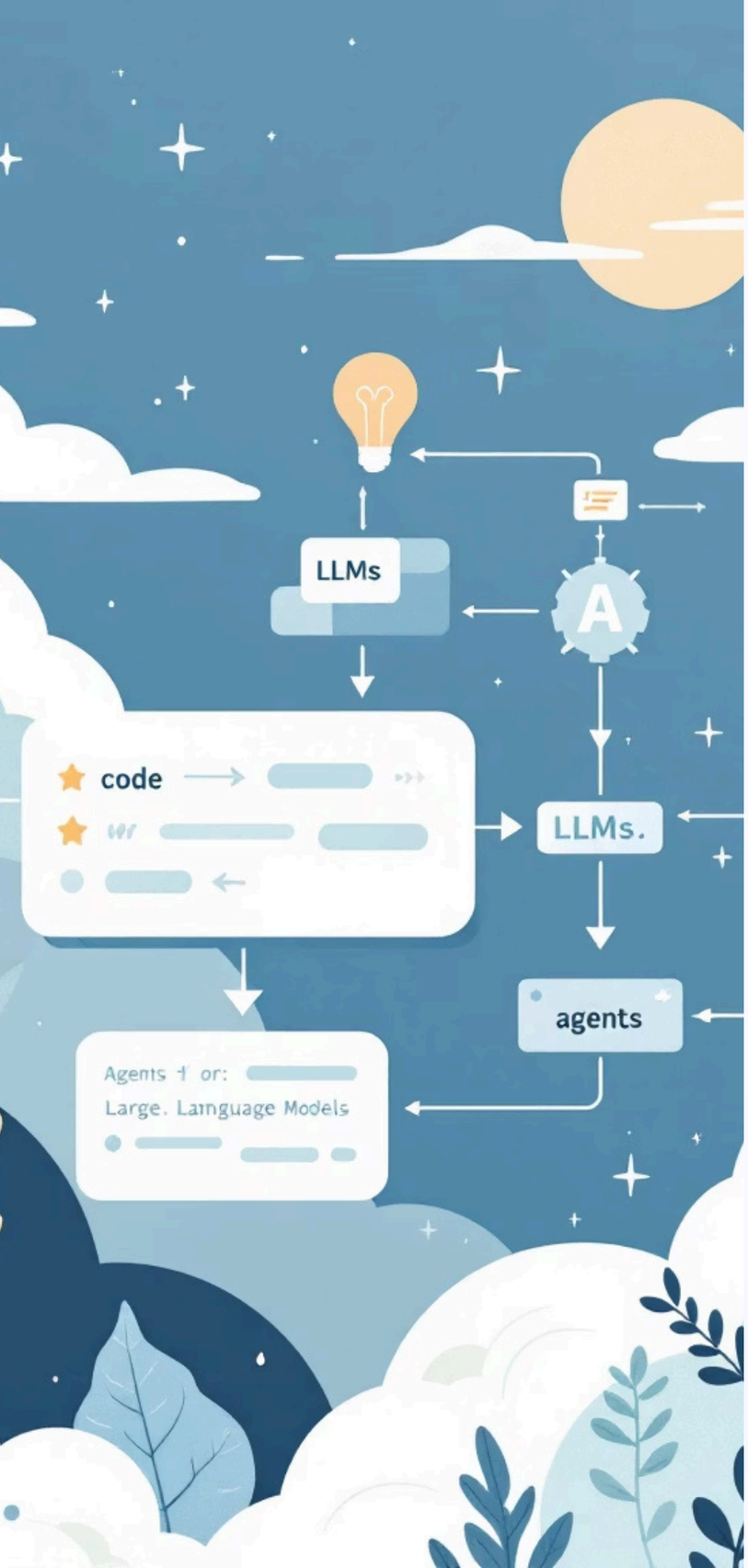
3 **Task Diversity**

Tasks span various difficulty levels and algorithmic concepts.

4 **Evaluation Ground Truth**

Unit tests serve as ground truth for correctness evaluation.





Evaluation Pipeline & Methodology

Our evaluation pipeline rigorously assesses five judge configurations against a unit test baseline, using 70 LeetCode-style tasks from LiveCodeBench. Each evaluator scores solutions on a 0-1 scale. We analyze from both numerical and binary perspectives.

01

Unit Test Baseline

- Ground truth for "correctness"
- Pass Rate [0,1]

02

LLM-as-a-Judge

- **Correctness: [0,1]**
- **Multi-aspect**:
  - correctness [0,1]
  - style [0,1]
  - simplicity [0,1]
  - robustness [0,1]

03

Agent-as-a-Judge

- **Correctness**
- **Multi-aspect**
- **Agent-Unit-Test**: direct instruction to write, exec, and inspect code files

04

Alignment Measurement

- **AI vs. Ground Truth UT**
- **Numeric & Binary**
- **Agent vs. LLM Judges**
- **Agent: Spontaneous vs. Directed Behavior**

Task Workflow Example: Minimum Operations to Make Median of Array Equal to K

01

PROBLEM: Minimum Operations to Make Median of Array Equal to K

Given an integer array nums and a non-negative integer k. In one operation, you can increase or decrease any element by 1. Return the minimum number of operations needed to make the median of nums equal to k.

02

SOLUTION: Sorting + Two-Pointer Adjustment

Sort the array, find the median position (n//2 for even length), make the median equal to k, then adjust elements left of median down to k if needed, and elements right of median up to k if needed.

03

UNIT TEST Baseline

All 28 tests PASSED - 7 public tests + 21 private tests covering various array sizes and k values.

04

LLM/Agent (Correctness-Only)

Score: 1.0

Correct approach: sort, pick upper-middle index, sum necessary adjustments left and right of median; handles odd/even lengths correctly; O(n log n) complexity.

05

LLM/Agent (Multi-Aspect)

- Correctness: 1.0
- Style: 0.6 (unnecessary imports, verbose comments)
- Simplicity: 0.7 (core logic straightforward but cluttered)
- Robustness: 0.8 (handles edge cases well, but in-place mutation reduces maintainability)

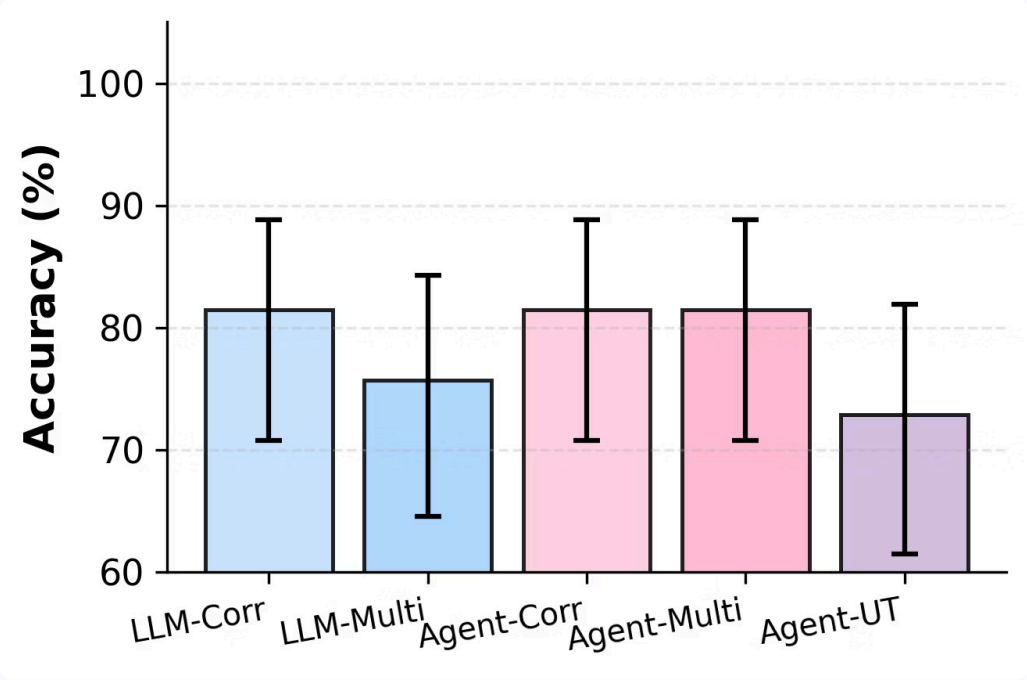
06

AGENT-UT (Test-Writing Agent)

- Correctness: 1.0
- Style: 0.3 (excessive imports, name collision risks)
- Simplicity: 0.8 (optimal algorithm, concise)
- Robustness: 0.8 (handles edge cases, scales well)

Note: Passed all 28 unit tests + 1000 randomized test cases

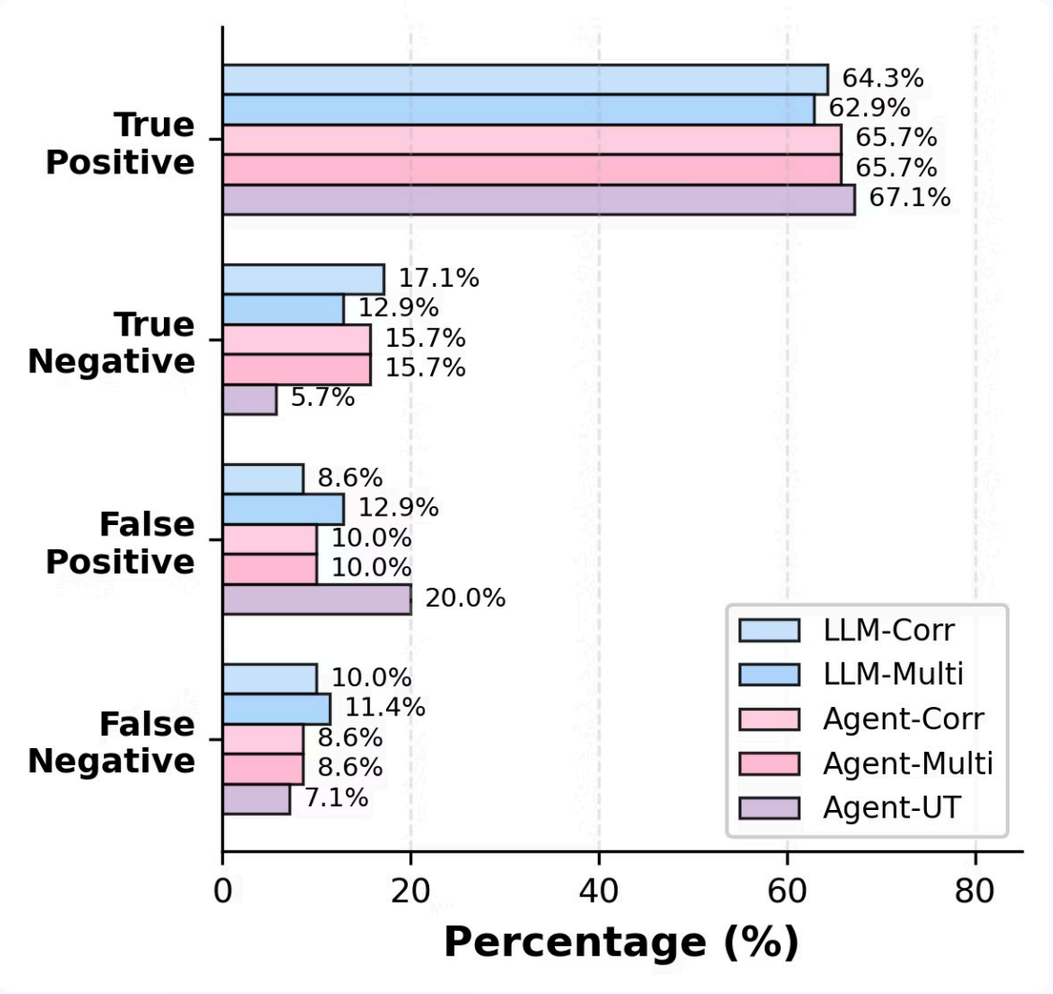
Results: Score Distribution & Binary Accuracy



Left: Binary accuracy ranges 72-82%, with LLM judges more consistent (80-81%) than Agent-UT (74%). Right: Score distributions reveal ground truth's clear bimodal pattern (0 or 1), while AI judges show systematic clustering–LLM judges spread across 0.2-0.8 range, Agent-UT tighter but shifted. Mean scores (purple diamonds) show conservative bias in most judges except Agent-UT.

Results: Binary Accuracy Confusion Matrix Bar

- LLM judges: 80-81% accuracy (consistent). Agent-UT: 74% accuracy (wider variance).



- LLM bias: high true positives, low true negatives (conservative). Agent-UT: more false positives (optimistic).

Results: Agent Testing Behavior & Cost

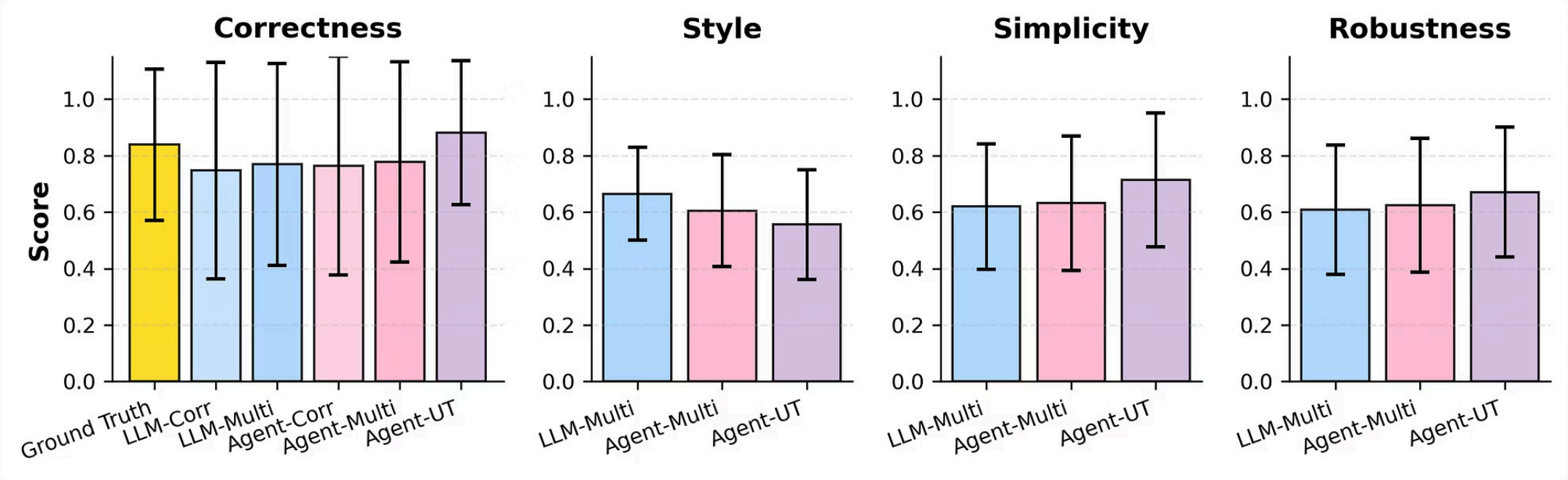
Agent Behavior Metrics: Steps and Spontaneous Testing

Agent Scenario	Number of Steps (Mean ± SD)	Spontaneous Unit-Tests (%)	N Tasks
Agent-Corr	13.4 (±18.1)	7.1%	70
Agent-Multi	10.4 (±5.3)	2.9%	70
Agent-UT	29.7 (±50.2)	100.0%	70

Spontaneous Unit-Tests: Percentage of instances where agent explicitly wrote and executed unit tests.
Agent-UT is designed to create tests (100%), while others may spontaneously test.

Spontaneous test writing is rare (2.9-7.1% of tasks). Agent-UT with explicit instructions writes tests for 100% of tasks but at 2-3x higher computational cost (29.7 ± 50.2 steps vs 10-13 steps). This demonstrates agents need explicit direction to leverage testing capabilities, with significant cost implications.

Results: Multi-Aspect Evaluation Scores



Correctness scores are consistently high (0.8-1.0) across all judges. However, style, simplicity, and robustness scores are systematically lower (0.4-0.7), revealing code quality issues beyond mere functional correctness. Robustness, in particular, shows the most variance, indicating AI judges can identify potential failures that unit tests might miss. This multi-dimensional evaluation highlights both the advantage (providing a richer, more holistic assessment) and the risk (potential false positives from subjective judgment) inherent in subjective code assessment.

```
Running command: python3 - << 'PY'
from itertools import product

def naive(n, k, s):
    cells = list(s)
    ops = 0
    i = 0
    while i < n:
        if cells[i] == 'B':
            # Erase from i to i+k
            for j in range(i, min(i+k, n)):
                cells[j] = 'W'
            ops += 1
            i += 1
        else:
            i += 1
    return ops

def optimal(n, k, s):
    blacks = [i for i, c in enumerate(s) if c == 'B']
    if not blacks:
        return 0
    ops = 0
    covered = set()
    i = 0

    uncovered = set(blacks)
    while uncovered:
        pos = min(uncovered)
        p = min(pos, n - k)

        for j in range(p, p + k):
            if j in uncovered:
                uncovered.remove(j)
        ops += 1
    return ops

counterexamples = []
for n in range(1, 9):
    for k in range(1, n + 1):
        all_strings = [''.join(chars) for chars in product('BW', repeat=n)]

        for s in all_strings:
            a = naive(n, k, s)
            b = optimal(n, k, s)

            if a != b:
                counterexamples.append((n, k, s, a, b))
                if len(counterexamples) < 20:
                    print('CE', n, k, s, a, b)

print('Total counterexamples found:', len(counterexamples))
```

Case Study 1: 1D Eraser

This case demonstrates an instance in which the agent autonomously generated test cases.

> Problem: You are given a strip of paper □ that is □ cells long. Each cell is either black or white. In an operation you can take any □ consecutive cells and make them all white. Find the minimum number of operations needed to remove all black cells.

> **Agent Multi-aspect score:**
>
> Correctness_Score: 1.0
>
> Style_Score: 0.5
>
> Simplicity_Score: 0.8
>
> Robustness_Score: 0.7

Spontaneous tests tend to increase the agent's confidence, and often increase its scores.

Case Study 2: Find the Number of Subsequences With Equal GCD

This case demonstrates an instance in which the LLM/Agent Judges Lower Than Unit Tests but with a valid reason.

Problem: You are given an integer array nums.

Your task is to find the number of pairs of non-empty subsequences (seq1, seq2) of nums that satisfy the following conditions:

The subsequences seq1 and seq2 are disjoint, meaning no index of nums is common between them. The GCD of the elements of seq1 is equal to the GCD of the elements of seq2. Return the total number of such pairs.

```python
class Solution:
    def subsequencePairCount(self, nums: List[int]) -> int:
        MOD = 10**9 + 7
        n = len(nums)

        # Use memoization with recursion
        # State: (index, gcd1, gcd2) where gcd1 is GCD of seq1, gcd2 i
        # 0 means empty sequence
        @lru_cache(maxsize=None)
        def dp(index, gcd1, gcd2):
            # Base case: processed all elements
            if index == n:
                # Both sequences must be non-empty and have equal GCD
                if gcd1 > 0 and gcd2 > 0 and gcd1 == gcd2:
                    return 1
                return 0

            result = 0
            current = nums[index]

            # Option 1: Add to seq1
            new_gcd1 = gcd(gcd1, current) if gcd1 > 0 else current
            result += dp(index + 1, new_gcd1, gcd2)
            result %= MOD

            # Option 2: Add to seq2
            new_gcd2 = gcd(gcd2, current) if gcd2 > 0 else current
            result += dp(index + 1, gcd1, new_gcd2)
            result %= MOD

            # Option 3: Skip (don't add to either sequence)
            result += dp(index + 1, gcd1, gcd2)
            result %= MOD

            return result

        return dp(0, 0, 0)
```

Multi-Aspect Agent Judge: Correctness: 0.3, Style: 0.4, Simplicity: 0.4, Robustness: 0.4 → Average: 0.375

Correctness-Only Agent Judge: Score: 0.3

Multi-Aspect LLM Judge: Correctness: 0.20, Style: 0.30, Simplicity: 0.20, Robustness: 0.20 → Average: 0.225

Correctness-Only LLM Judge: Score: 0.23

Unit Test: Score: 1.00 (43/43 tests passed - PASSED)

Multi-Aspect Agent Judge:

"The algorithm is logically correct (it enumerates all assignments of elements to seq1, seq2, or neither) but is exponential in practice and will likely TLE or MLE for the maximum constraints (n up to 200), so it won't pass large test cases."

LLM Correctness-Only Judge:

"The recursive DP correctly models ordered disjoint subsequence assignments and counts pairs with equal non-empty GCDs, so it will pass small tests and the provided examples. However, with n up to 200 and gcd states up to ~200×200 per index, the lru_cache will explode in time/memory and will TLE/MLE on large worst-case tests, so it will fail most heavy cases."

LLM/Agent judges can be better than Unit tests at identifying theoretical performance bottlenecks, leading to valid lower scores even when all provided tests pass.

Key Insights & Findings (based on OpenHands + GPT-5-mini)

Partial Alignment but good alternative

AI judges preserve most correctness structure but with certain disagreement.

Conservative vs Optimistic

Non-test-writing judges underestimate (conservative); test-writing agents overestimate (less comprehensive / coverage).

Subjectivity Trade-off

Multi-aspect evaluation reveals issues beyond unit tests but introduces noise.

Cost-Fidelity Spectrum

From cheap LLM judges (lower fidelity) to expensive Agent-UT (higher fidelity).

Agent Capabilities

Agents need explicit instructions to leverage testing; spontaneous testing exhibits but is rare.

Q&A

CS5787 Final Project

Comparing **LLM Judges**, **Agent Judges**, and **Unit Tests**

Total Tasks: **67**

1873_d__ewd5mxi__x9Nv7BA

D. 1D Eraser

You are given a strip of paper s that is n cells long. Each cell is either black or white. In an operation you can take any k consecutive cells ...

1.00	0.70	0.80
UNIT TEST	LLM JUDGE	AGENT JUDGE

2808__j8qegjl__ChgukeA

painting-the-walls

You are given two 0-indexed integer arrays, cost and time, of size n representing the costs and the time taken to paint n different walls respectively...

1.00	0.75	0.80
UNIT TEST	LLM JUDGE	AGENT JUDGE

2848__8ofyxkj__3hXEi4v

special-permutations

You are given a 0-indexed integer array nums containing n distinct positive integers. A permutation of nums is called special if: For all indexes 0 <=...

1.00	0.70	1.00
UNIT TEST	LLM JUDGE	AGENT JUDGE

2850__qap9g7a__mFMciSk

construct-the-longest-new-string

You are given three integers x, y, and z. You have x strings equal to "AA", y strings equal to "BB", and z strings equal to "AB". You want to choose s...

0.64	0.38	0.80
UNIT TEST	LLM JUDGE	AGENT JUDGE

2955__jzk6brf__LJUAFWR

account-balance-after-rounded-purchase

Initially, you have a bank account balance of 100 dollars. You are given an integer purchaseAmount representing the amount you will spend on a purchas...

1.00	0.75	1.00
UNIT TEST	LLM JUDGE	AGENT JUDGE

3034__vwyjl7t__5tEhgNf

points-that-intersect-with-cars

You are given a 0-indexed 2D integer array nums representing the coordinates of the cars parking on a number line. For any index i, nums i = [start_i...

1.00	0.82	1.00
UNIT TEST	LLM JUDGE	AGENT JUDGE

3046__yzlidf9__3gffzPS

minimum-operations-to-make-a-special-number

You are given a 0-indexed string num representing a non-negative integer. In one operation, you can pick any digit of num and delete it. Note that if ...

1.00	0.80	0.80
UNIT TEST	LLM JUDGE	AGENT JUDGE

3163__ptpmytw__saM6YkA

subarrays-distinct-element-sum-of-squares-i

You are given a 0-indexed integer array nums. The distinct count of a subarray of nums is defined as: Let nums $i..j$ be a subarray of nums consisting ...

1.00	0.70	1.00
UNIT TEST	LLM JUDGE	AGENT JUDGE

3193__og74rb4__DyDE8mx

maximum-strong-pair-xor-i

You are given a 0-indexed integer array nums. A pair of integers x and y is called a strong pair if it satisfies the condition: |x - y| <= min x, y Y...

1.00	0.78	1.00
UNIT TEST	LLM JUDGE	AGENT JUDGE