EVALUATION METRICS

Metric	Definition	Value ranges	System's Value	Interpretation	Verdict
Problem Success Rate	Fraction of problems for which the system successfully completes: 1. formal specification 2. test generation 3. reference solution construction 4. golden test suite generation	0.0 → complete failure 1.0 → end-to-end success on all problems	**0.71**	~71 out of 100 vague problems successfully completed the full pipeline. ~29% failed due to: ● specification ambiguity, ● invalid test generation, ● brute/optimized solution disagreement, ● or circular healing exhaustion.	**Moderate, not excellent** For a system operating entirely on **LLM reasoning with no human curation**, this is **acceptable but clearly improvable.**
Test Survival Rate	Fraction of generated candidate tests that survive into the final golden suite.	0.0 → all tests rejected 1.0 → no tests discarded	**0.9749**	Only ~2.5% of generated tests are discarded. Indicates: ● strong constraint adherence, ● effective test validation, ● minimal over-generation of invalid inputs.	**Excellent**. The **test oracle is disciplined and conservative**,
Consensus Converg	Fraction of successful problems where	0.0 → no agreement	**0.9718**	When the pipeline succeeds, **~97% of problems reach**	**Very strong**

ence Rate	optimized and brute-force solutions converge to consistent outputs.	1.0 → full agreement		**consensus**. Disagreements are rare and typically resolved via healing.	Shows that treating brute force as “truth” is operationally effective.
Circular Healing Frequency	Fraction of successful problems where circular healing (Stage 5 rejecting Stage 4 tests) was triggered.	0.0 → never needed Higher is worse	**0.0563**	Circular healing was triggered in ~5.6% of successful runs. Indicates occasional mismatch between: ● test generator assumptions, and ● reference solution feasibility.	**Low but non-zero** The system **catches its own mistakes**, which is far better than silently accepting bad tests.
Healing Yield	Fraction of healing attempts that successfully corrected the optimized solution	0.0 → healing useless 1.0 → healing always succeeds	**0.4348**	Only ~43% of healing attempts successfully fix errors. More than half fail or stall.	Weak. LLM self-correction is **unreliable, BUT** with user injected counterexamples, the reference solutions would increase the yield multi-folds.

LATENCY ANALYSIS

Stage	Avg Latency (s)	Assessment
Stage 3	3.29	Reasonable
Stage 4	9.50	Heavy but expected
Stage 5	13.92	Expensive
Stage 5.2	7.09	Moderate
End-to-end	37.37	Slow but acceptable, because the user receives real-time feedback on which

		stage our model is in.

GOLD BENCHMARK - SYSTEM SOLUTION CORRECTNESS

Verdict: 12/15 unknown problems met the ground-truth correctness.

Vague Query	Persona	Analysis	Verdict	Question ID
Python solution to fix a binary search tree by identifying and swapping the values of the two incorrect nodes back to their correct positions. Input: root = [1,3,null,null,2] Output: [3,1,null,null,2]	implementation_specific	Test suite failure. Can't create a binary tree that's just two nodes away from a BST. Good syntax understanding, but lacks semantics!	Failed	1
Input: T: binary search tree. Output: T': corrected binary search tree with original node values restored. There are exactly two node values misplaced. Input: root = [1,3,null,null,2] Output: [3,1,null,null,2]	technical_shorthand	Test suite failure. Can't create a binary tree that's just two nodes away from a BST. Good syntax understanding, but lacks semantics!	Failed	1
Input: $nums: array$. Output: int. Count continuous subarrays where $\|nums[i1] - nums[i2]\| <= 2$ for all $i1, i2$ in the subarray. Input: nums = [5,4,2,4] Output: 8	technical_shorthand	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 0	Passed	2
Python solution to count the number of continuous segments in an array where the difference between the largest and smallest element is 2 or less. Input: nums = [5,4,2,4] Output: 8	implementation_specific	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 0	Passed	2
Input: $T: array$ of characters, $n: integer$. Output: int.	technical_shorthand	Formal Specification: Correct	Passed	3

Return minimum intervals required to complete all tasks with at least n intervals between identical tasks. Input: tasks = ["A","A","A","B","B","B"], n = 2 Output: 8		# Attempts: 1 # Circular Heals: 1 # Test cycles: 1/1 # Solution cycles: 3/1 # User feedback: 0		
Python implementation to calculate the minimum number of intervals needed to finish a list of CPU tasks, ensuring a gap of at least 'n' intervals between tasks with the same label. Input: tasks = ["A","A","A","B","B","B"], n = 2 Output: 8	implementation_specific	Formal Specification: Correct # Attempts: 1 # Circular Heals: 3 # Test cycles: 1/1/1 # Solution cycles: 3/3/1 # User feedback: 0	Passed	3
Input: $S: string$. Output: int. Return maximum product of lengths of two disjoint palindromic subsequences of S. Input: s = "leetcodecom" Output: 9	technical_shorthand	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 0	Passed	4
Python code to find the maximum product of lengths of two non-overlapping palindromic subsequences in a given string. Input: s = "leetcodecom" Output: 9	implementation_specific	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 1 (At first gave TLE, but upon user feedback, gave more optimized example)	Passed	4
Input: $S: string$. Output: $List$ of $substrings$. Return all 10-letter-long $substrings$ of S that occur more than once. Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT" Output: ["AAAAACCCCC","CCCCCAAAAA"]	technical_shorthand	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 2 # Solution cycles: 1 # User feedback: 0	Passed	5
Python solution to identify all repeated 10-letter sequences in a given DNA string. Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT" Output: ["AAAAACCCCC","CCCCCAAAA	implementation_specific	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 2 # Solution cycles: 1	Passed	5

A"]		# User feedback: 0		
Input: $edges: array$, $distanceThreshold: int$. Output: $city: int$. Return city with the least reachable cities under distanceThreshold, break ties by choosing the larger city number. Input: n = 4, edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]], distanceThreshold = 4 Output: 3	technical_short hand	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 0	Passed	6
Python code to determine the city with the fewest reachable neighbors within a given distance, returning the largest city number in case of ties. Input: n = 4, edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]], distanceThreshold = 4 Output: 3	implementation _specific	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 0	Passed	6
Input: $nums: array$, $k: int$. Output: int. Return the minimum number of k-bit flips to make all elements in $nums$ equal to 1, or -1 if not possible. Input: nums = [0,1,0], k = 1 Output: 2	technical_short hand	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 0	Passed	7
Python solution to calculate the least number of flips needed to turn an array of 0s and 1s into all 1s by flipping segments of length k. Input: nums = [0,1,0], k = 1 Output: 2	implementation _specific	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 0	Passed	7
Input: $T: binary tree$, $L: linked list$. Output: $bool$. Return $True$ if path from T matches all elements of L from head downwards. Input: head = [4,2,8], root = [1,4,4,null,2,2,null,1,null,6,8,null,null,null,null,1,3] Output: true	technical_short hand	Formal Specification: Wrong # Attempts: 3 # Circular Heals: 3 # Test cycles: 3/3/3 # Solution cycles: 3/3/3 # User feedback: 0 DIFFICULTY IN INTERPRETING THE	Failed	8

		LINKED-LIST and BINARY TREE INPUTS		
Python solution to verify if a linked list matches a downward path in a binary tree, starting from the tree's nodes. **Input: head = [4,2,8], root = [1,4,4,null,2,2,null,1,null,6,8,null,null,null,1,3]** **Output: true**	**implementation_specific**	**Formal Specification: Wrong** **# Attempts: 10** **DIFFICULTY IN INTERPRETING THE LINKED-LIST and BINARY TREE INPUTS**	**Failed - Took me 8 attempts**	**8**
Input: $nums: array of integers$. **Output: int. Count permutations of $nums$ such that $nums[i] + nums[i+1]$ is a perfect square for all valid i.** **Input: nums = [1,17,8]** **Output: 2**	**technical_shorthand**	**Formal Specification: Correct** **# Attempts: 1** **# Circular Heals: 0** **# Test cycles: 1** **# Solution cycles: 1** **# User feedback: 1 (Pass HINT as solution to Strong Prior Bias)** **Suffers from GPT's Strong Prior Bias. 'Solution' object has no attribute 'solve'**	**Passed**	**9**
Python implementation to count distinct arrangements of an integer array where the sum of each adjacent pair results in a perfect square. **Input: nums = [1,17,8] Output: 2**	**implementation_specific**	**Formal Specification: Correct** **# Attempts: 1** **# Circular Heals: 0** **# Test cycles: 1** **# Solution cycles: 1** **# User feedback: 1 (Pass HINT as solution to Strong Prior Bias)** **Suffers from GPT's Strong Prior Bias. 'Solution' object has no attribute 'solve'**	**Passed**	**9**
You are given the root of a binary tree. We install cameras on the tree nodes where each camera at a node can monitor its parent, itself, and its immediate children. Return the minimum number of cameras needed to monitor all nodes of the tree.	**Formal (Leetcode)**	**Formal Specification: Correct** **# Attempts: 1** **# Circular Heals: 0** **# Test cycles: 2** **# Solution cycles: 1** **# User feedback: 0**	**Passed**	**10**

Input: root = [0,0,null,0,0] Output: 1				
A binary tree is uni-valued if every node in the tree has the same value. Given the root of a binary tree, return true if the given tree is uni-valued, or false otherwise. Input: root = [1,1,1,1,1,null,1] Output: true	Formal (Leetcode)	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 2 # Solution cycles: 1 # User feedback: 0	Passed	11
Input: $nums: array$. Output: int. Return minimum swaps to group all 1's in $nums$ considering it as a circular array. Input: nums = [0,1,0,1,1,0,0] Output: 1	technical_shorthand	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 0	Passed	12
Python solution to count land cells in a binary matrix that are completely enclosed by water, meaning they cannot reach the grid's edge.	implementation_specific	Formal Specification: Correct # Attempts: 1 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 0	Passed	13
Python solution to manage two lists where you can flip values in the first list based on commands, update the second list using values from the first list multiplied by a given number, and finally calculate the total of the second list after all operations.	implementation_specific	Formal Specification: Wrong # Attempts: 2	Failed	14
Python implementation to group integers into pairs from an array and calculate the maximum sum of the minimums of those pairs. Input: nums = [1,4,3,2] Output: 4	implementation_specific	Formal Specification: Correct # Attempts: 2 # Circular Heals: 0 # Test cycles: 1 # Solution cycles: 1 # User feedback: 0	Passed	15