1. Summary

We evaluated **GPT-4o mini** on the LiveCodeBench dataset. The model failed **46/100 problems**, distributed across four high-level failure categories:

Failure Category	Problem Count
Wrong Answer	32
Time Limit Exceeded (TLE)	6
Runtime Error	2
Combination of Failures	6
Total	**46**

Primary finding: logical correctness and constraint interpretation account for the majority (~70%) of failures. Efficiency and stability are secondary issues.

2. Test methodology (summary of assumptions)

- Evaluation used the LiveCodeBench problems.
- Each problem's result was categorized by the observable failure mode: Wrong Answer (WA), Time Limit Exceeded (TLE), Runtime Error (RE), or Combination of Failures (multiple attempt outcomes).

3. High-level failure breakdown (keeps original reasoning intact)

3.1 Wrong Answer — 32 problems

These solutions passed initial checks but produced incorrect output for at least one test case. The underlying reasons are broken down into three subcategories:

a) Flawed or Incomplete Logic

The algorithm was fundamentally incorrect and did not address the core logic of the problem.

- **C. Raspberries:** The logic oversimplified the divisibility problem, failing to account for how multiple numbers in a product can satisfy the condition.

- **B. 250 Thousand Tons of TNT:** The solutions incorrectly sorted the array of weights, which destroyed the circular and contiguous nature of the box arrangement described in the problem.

- **Yarik and Array:** Failing solutions, often based on Kadane's algorithm, did not correctly reset the subarray sum when two adjacent elements had the same parity.

- **Yarik and Musical Notes:** The solutions failed to account for all mathematical conditions where bibj=bjbib_i^{b_j} = b_j^{b_i}bibj=bjbi, specifically missing the special case of (2, 4).

- **maximum-or:** The solutions failed to iterate through each number as the target for the kkk multiplications, instead applying them to a single, arbitrarily chosen number.

- **neighboring-bitwise-xor:** The logic for reconstructing the original binary array from the XOR of its neighbors was incorrect.

- **find-the-string-with-lcp:** The solutions could not handle the complex logic of assigning characters while satisfying all Longest Common Prefix (LCP) matrix constraints.

- **sum-of-matrix-after-queries:** The logic failed to correctly handle the overriding nature of the queries, leading to double-counting or incorrect final values.

b) Misinterpretation of Problem Constraints

The solution addressed a problem that was slightly different from the one described due to a misunderstanding of the rules or goals.

- **A. Game with Integers:** A simple game theory problem where the winning condition was inverted. The solution incorrectly identified the winner based on the starting number's divisibility by 3.

- **shortest-string-that-contains-three-strings:** The solutions failed to check all possible permutations of merging the three strings to find the absolute shortest superstring.

- **find-the-punishment-number-of-an-integer:** The recursive logic for partitioning the square of a number and summing the parts was flawed, indicating a misunderstanding of the partitioning rules.

- **make-costs-of-paths-equal-in-a-binary-tree:** The logic for calculating the cost adjustments to equalize path sums was incorrect.

c) Poor Edge Case Handling

The algorithm was mostly correct but failed on specific edge cases.

- **B. Chemistry:** Solutions were often off-by-one in their logic, failing when the number of characters with odd frequencies was close to kkk.
- **maximum-strength-of-a-group:** The solutions failed to handle cases involving zeros or when the optimal answer was a single negative number (e.g., for an input of [0][0][0], one solution returned 1 instead of 0).
- **extra-characters-in-a-string:** The dynamic programming solutions had incorrect base cases or flawed state transitions, causing them to fail on certain string/dictionary combinations.
- **check-if-it-is-possible-to-split-array:** The recursive solutions did not correctly handle base cases like single-element arrays, leading to errors or wrong answers.

3.2 Time Limit Exceeded (TLE) — 6 problems

These solutions were too inefficient for large test cases, pointing to suboptimal algorithmic complexity or use of brute force where optimized approaches were required.

- **count-of-integers**
- **find-the-minimum-possible-sum-of-a-beautiful-array**
- **number-of-ways-to-build-a-pyramid**
- **count-the-number-of-powerful-integers**
- **find-the-number-of-ways-to-place-people-ii**
- **find-the-sum-of-the-power-of-all-subsequences**

Typical root cause: produced \`O(n^2)\`, \`O(n^3)\`, or combinatorial solutions when \`O(n log n)\`, greedy, or mathematical reductions were required.

3.3 Runtime Error — 2 problems

The program crashed during execution due to unhandled exceptions or incorrect usage of language primitives.

- **Short Sort:** Failed due to a **TypeError**, indicating a type mismatch in an operation (e.g., using a string as a list index).
- **sum-in-a-matrix:** Failed due to a **ValueError**, caused by attempting to remove an element from a list that wasn't present or calling `max()` on an empty list.

Typical root cause: insufficient input validation, missing guards for empty inputs, or incorrect type assumptions.

3.4 Combination of Failures — 6 problems

Different solution attempts failed for different reasons, highlighting problem difficulty and variability across generated solutions.

- **Yarik and Musical Notes:** Some attempts had flawed logic ("Wrong Answer"), while others were too slow ("Time Limit Exceeded").
- **maximize-the-number-of-partitions-after-operations:** Failures due to both incorrect logic and inefficient solutions.
- **count-the-number-of-complete-components:** Failures included "Wrong Answer" and "Time Limit Exceeded".
- **frequency-tracker:** Failures included "Wrong Answer" and "Time Limit Exceeded".
- **find-the-longest-semi-repetitive-substring:** Failures included "Wrong Answer" and "Time Limit Exceeded".
- **find-beautiful-indices-in-the-given-array-i:** Failures included "Wrong Answer" and "Time Limit Exceeded".

These mixed outcomes indicate instability in both algorithm selection and correctness across different sampling seeds/prompts.

---

4. Key insights (preserving your reasoning)

1. **Logical gaps dominate failures.** The majority of failing cases were due to incorrect or incomplete problem logic rather than syntax issues.
2. **Subtle constraint misinterpretation is common.** The model often solves a “nearby” problem — close but different — suggesting lapses in strict reading/comprehension of constraints.
3. **Edge cases expose fragile reasoning.** Solutions that look correct on surface tests often fail on corner inputs (zeros, minimal lengths, special numeric pairs).
4. **Efficiency is not consistently handled.** GPT-4o mini often returns straightforward or brute-force approaches when optimized algorithms are necessary for large inputs.
5. **Execution stability is relatively strong.** Only a small fraction of failures were runtime errors, indicating syntactic competence but weaker algorithmic correctness.

9. Conclusion

- GPT-4o mini demonstrates decent surface-level code generation ability (low RE rate), but **reasoning completeness, constraint fidelity, and algorithm selection** are the main failure vectors on LiveCodeBench.