Evaluation Report: GPT■4o Mini on LiveCodeBench

This report provides a detailed analysis of GPT■4o Mini's performance on the LiveCodeBench benchmark. Out of the problems attempted, **46 problems failed** across multiple categories, with failures distributed among Wrong Answers, Time Limit Exceeded (TLE), Runtime Errors, and Combination Failures. The following sections provide a breakdown of these categories and the specific reasoning for the failures.

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

1. Wrong Answer

These solutions passed the initial checks but produced incorrect output for at least one test case. The underlying reasons can be broken down further:

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. - **C. 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. - **D. Yarik and Musical Notes**: The solutions failed to account for all mathematical conditions where b_i^b_j = b_j^b_i, specifically missing the special case of (2, 4). - **maximum-or**: The solutions failed to iterate through each number as the target for the k 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 k. - **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], 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.

2. Time Limit Exceeded (TLE)

These solutions were too inefficient and could not pass larger test cases within the given time constraints. This typically points to a suboptimal algorithmic complexity (e.g., brute-force where a more optimized approach was needed). - 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

3. Runtime Error

These solutions crashed during execution. - **A. 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.

4. Combination of Failures

For these problems, different solution attempts failed for different reasons, highlighting the difficulty of the problem. - **D. 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**: Similar to the above, failures were 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".