Golden Code Response & Golden Unit Tests

The first step before actually attempting to write the golden code & unit tests is evaluating the seeded unit tests against the five criteria.

Five Golden Unit Test Criteria

#	Criteria	Description
1	**High Value Code Coverage**	Do the tests cover core logic and critical paths, not just boilerplate?
2	**Meaningful Assertions**	Does each test verify specific, important behavior, not just that the code runs?
3	**Edge Case & Input Variety**	Does the unit test suite include tests for nulls, empty data, min/max values, invalid inputs, etc?
4	**Test Independence**	Do the tests not rely on shared state or order of execution?
5	**No Flaky or Redundant Tests**	Do the tests always pass/fail deterministically? Are there any duplicates doing the same check?

Step 1: Unit Test Evaluation

Action	Description
Evaluate Against Five Criteria	Assess current unit test quality using standardized benchmarks
Mark Yes/No	For each of the five criteria, mark a definitive "Yes" or "No" based on your evaluation. If marking "No" for any criterion, document specific examples of why the tests fail to meet that standard.

Action	Description
	Note that ANY "No" marking requires you to rewrite the unit tests in the ‘test’ file during the next stage. Prepare detailed notes on what changes are needed to transform each "No" into a "Yes".

Step 2: Golden Unit Test Creation

#	Step	Description
1	**Access The Second Workspace (Editable)**	Navigate to the second workspace by clicking the blue [Open IDE] button. This environment is fully editable, unlike the read-only first workspace.
2	**Transform Seeded Tests Into Golden Response**	Open the ‘test’ file that contains the original seeded unit tests. Using your detailed findings from the unit test evaluation. Focus on rewriting each test that failed the five criteria. Then add more unit tests, ensuring all critical functionality is well-tested and edge cases are covered. **Unit tests should not have trivial code or test tautologies (See examples)**
3	**Maintain Skeleton Function Focus**	Strictly limit your testing to the main functions **explicitly** defined in the skeleton code with their original function signatures While your final implementation may include additional helper functions for internal logic, the golden unit test suite must only validate the public interface functions provided in the skeleton.
4	**Final Validation**	• Conduct a thorough review of your modified ‘test’ file against each of the five criteria. • Run the test suite to ensure all tests execute properly and validate expected behaviors. • Check for any remaining gaps in coverage, weak assertions, missing edge cases, test dependencies, or redundant/flaky tests. • Make final adjustments to guarantee your test suite definitely scores "Yes" on all five criteria. Document your validation process for quality assurance.

Golden Code Generation

Action	Detailed Process
Generate LLM Response via Fabricator	Prepare a complete fabricator prompt for Deepseek R1 0528 that includes all six essential components: (1) Paraphrased Prompt providing a clear task description (2) Skeleton Code with function signatures to implement (3) Seeded Response showing the original failing attempt (4) Golden Unit Test Suite from your Stage 5 work (5) Failure Output demonstrating specific errors when seeded response runs against golden tests (6) Model Guidance providing technical direction for fixes Submit this comprehensive prompt to generate an initial corrected implementation that will only be used as **A STARTING POINT** for your golden response.
Code Extraction & Initial Testing	Extract only the pure code implementation from the Deepseek R1 0528 response, removing any explanatory text, markdown formatting, or extraneous content. • Paste this code into your workspace and execute it against your golden unit test suite. • Carefully observe all test results, noting any failures, errors, or unexpected behaviors. If you discover issues with your golden test suite during this process (such as incorrect expected values or flawed test logic), immediately fix these problems directly in the 'test' file before proceeding.
Iterative Refinement Process	If the extracted code fails any golden unit tests, analyze the specific failure points and refine your approach, then reiterate. • This may involve adjusting the fabricator prompt to provide clearer guidance, adding more specific technical details about the failures, or restructuring the model guidance to better address the root causes. • Regenerate the response with Deepseek R1 0528 using the improved prompt until you achieve a version that passes all golden unit tests without exceptions.

Action	Detailed Process
	If you don't get a good response after more than 3 to 5 tries, it's recommended to grab the current code it returned (if helpful) and rewrite it yourself into a golden code response.
STOP Validate The Golden Unit Test Suite Are Actually Golden	Validate that the golden unit test suite is golden. • **A common mistake is not spending enough time writing a golden unit test suite and having key gaps/core functionality not being tested** • The moment you truly have a golden unit test suite, testing your golden response is straightforward.
🚨CRUCIAL Substantial Code Enhancement & Major Code Refactoring🚨	Use the passing Deepseek R1 0528 response as your foundation, but implement significant modifications and improvements. • This is mandatory - your final code cannot be identical to the generated response. Enhance code structure, improve variable naming, add comprehensive error handling, optimize algorithms for better performance, ~~include detailed comments explaining complex logic,~~ remove comments, and implement additional robustness features. • Keep function signatures intact when refactoring • Make sure that the final submission is your original work while using Deepseek R1 0528 as your starting point.
Finalize Golden Implementation	**Develop your final golden code response that represents the highest quality implementation possible.** • Ensure all skeleton code functions are fully implemented with correct logic and appropriate helper functions are included as needed for maintainability. • The code should demonstrate excellent programming practices, including efficient algorithms, proper error handling, and maintainable structure. • Verify one final time that this golden implementation passes 100% of your golden unit test suite and fully satisfies all original prompt requirements.

Example Factory Prompt

Context:
You're given a prompt, skeleton code, unit tests, a failing model response, and test failure output.
Your task is to analyze the failures and provide a correct implementation.

Instructions:

1. Analyze the failing model response to understand what went wrong
2. Review the failure output to identify specific issues (syntax errors, logic errors, type mismatches, etc.)
3. Implement the skeleton code functions correctly
4. Add any necessary helper functions
5. Ensure your solution addresses all identified failure points
6. Provide clean, efficient code that will now pass the failing unit tests

Prompt:
[PARAPHRASED_PROMPT]

Skeleton Code:
[SEEDED_SKELETON_CODE]

Failing Model Response:
[SEEDED_CODE_RESPONSE]

Golden Unit Tests:
[FINAL_GOLDEN_UNIT_TESTS]

Failure Output:
[OUTPUT_SEEDED_RESPONSE_AGAINST_FINAL_GOLDEN_UNIT_TESTS]

Final Note: Please provide the complete, corrected implementation, maintaining the function signatures of the skeleton code but also adding helper functions where necessary.