

Test Report

Run ID: 21097912899-py3.12 • Generated: 2026-01-17 17:15:47 • Duration: 286.46s

Plugin: v0.1.0 (2f498263985a34902252c53c11fb820445bd8f21) [dirty]

Repo: v0.1.1 (3e1297d90afc9d2cdf92f2e93b2e048a94409310)

LLM: ollama / llama3.2:1b (minimal context, 386 annotated)

91.09%

Total Coverage

<div>387</div> <div>TOTAL TESTS</div>	<div>387</div> <div>PASSED</div>	<div>0</div> <div>FAILED</div>	<div>0</div> <div>SKIPPED</div>
<div>0</div> <div>XFAILED</div>	<div>0</div> <div>XPASSED</div>	<div>0</div> <div>ERRORS</div>	

AI ASSESSMENT

Scenario: Test the aggregation of all policy when an aggregate directory is set.

Why Needed: This test prevents a regression where aggregating all policies would result in no tests being reported.

Key Assertions:

- The aggregated report should contain both retained tests.
- The length of the aggregated report should be equal to the number of retained tests.
- Each retained test should have an outcome of 'passed'.
- All retained tests should be included in the aggregated report.
- The aggregate directory is set before aggregating reports.
- A temporary directory is created for each run.
- Reports are written to a file in the temporary directory with a unique filename.
- The aggregated result is not None.
- Both retained tests have an outcome of 'passed'.

COVERAGE

src/pytest_llm_report/aggregation.py	69 lines (ranges: 52, 55-56, 59, 61-63, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 130-131, 134, 141, 146, 148-153, 155, 157-159, 170, 217, 219-223, 235, 245, 248-249, 251, 253, 276-279, 281)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies that the aggregate function returns None when the directory does not exist.

Why Needed: Prevents a potential bug where the aggregate function fails to handle cases where the aggregation directory does not exist.

Key Assertions:

- The `aggregate` method should return `None` when the specified directory does not exist.
- The `aggregate` method should raise an exception or handle the error in some way when the directory does not exist.
- The test should verify that the aggregate function behaves correctly even if the directory is missing or inaccessible.
- The test should check for any specific error messages or behavior when the directory does not exist.
- The `aggregate` method should be able to handle cases where the directory exists but is empty or has no files.
- The test should verify that the aggregate function can correctly aggregate data from a non-existent directory.

COVERAGE

src/pytest_llm_report/aggregation.py	7 lines (ranges: 52, 55-57, 109-111)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `aggregate` function picks the latest policy for aggregation.

Why Needed: This test prevents regression where the `aggregate` function fails to pick the correct policy when there are multiple reports with different outcomes in a single run.

Key Assertions:

- The outcome of the aggregated report is 'passed' (latest).
- There is only one test in the aggregated result.
- The aggregated run meta contains `is_aggregated=True` and `run_count=2`.
- The aggregated summary contains `passed=1` and `failed=0`.

COVERAGE

src/pytest_llm_report/aggregation.py	77 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 130-131, 134, 141, 146, 148-153, 155, 157-159, 170, 182, 184-188, 190-191, 194, 217, 219-223, 235, 245, 248-249, 251, 253, 276-279, 281)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The aggregator function should not be able to aggregate without a specified directory.

Why Needed: This test prevents a potential bug where the aggregator function fails to aggregate due to missing configuration.

Key Assertions:

- `agg.aggregate()` is `None`
- `mock_config.aggregate_dir` is `None`
- `agg.aggregate()` does not raise an exception when called with `None` as directory
- `mock_config.aggregate_dir` is set to `None` before calling `agg.aggregate()`
- `agg.aggregate()` should be able to aggregate without a specified directory

COVERAGE

<code>src/pytest_llm_report/aggregation.py</code>	3 lines (ranges: 44, 52-53)
<code>src/pytest_llm_report/collector.py</code>	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
<code>src/pytest_llm_report/plugin.py</code>	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The `aggregate` function should not produce any reports when no files are found.

Why Needed: This test prevents a regression where the `aggregate` function would incorrectly report that there were no aggregations performed.

Key Assertions:

- aggregator.aggregate() is None
- pathlib.Path.exists() returns True
- pathlib.Path.glob() returns an empty list

COVERAGE

src/pytest_llm_report/aggregation.py

9 lines (ranges: 52, 55-57, 109-110, 113-114, 170)

src/pytest_llm_report/collector.py

14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)

src/pytest_llm_report/plugin.py

6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that coverage and LLM annotations are properly deserialized and can be re-serialized after fix.

Why Needed: Prevents regression in core functionality by ensuring correct deserialization of LLM annotations.

Key Assertions:

- Coverage was correctly deserialized from the report.
- LLM annotation was correctly deserialized from the test case.
- The aggregated result can be re-serialized successfully with the updated LLM annotations.
- The coverage and LLM annotations were properly included in the serialized output.
- The confidence level of the LLM annotation was correctly set to 0.95.
- The scenario, why_needed, and key_assertions of the LLM annotation were correctly updated after deserialization.
- The test case's file path and line ranges were correctly preserved in the serialized output.
- The coverage entry for the module was correctly created with the specified file path and line ranges.

COVERAGE

src/pytest_llm_report/aggregation.py	81 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 130-131, 134-137, 141-144, 146, 148-153, 155, 157-159, 170, 182, 184-188, 194, 217, 219-223, 235, 245, 248-249, 251, 253, 276-279, 281)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	32 lines (ranges: 40-43, 104-107, 109-111, 113, 115, 161-165, 167, 169, 171, 173, 176-180, 182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: test_aggregate_with_source_coverage verifies that the source coverage summary is deserialized correctly.

Why Needed: This test prevents regression in handling JSON reports with multiple files and source coverage data.

Key Assertions:

- The `source_coverage` key should be present in each report.
- Each `source_coverage` value should have the required keys (file_path, statements, missed, covered, coverage_percent, covered_ranges, missed_ranges).
- All values in the `source_coverage` dictionary should be of type int or float.
- The `covered_ranges` and `missed_ranges` values should match the expected formats.
- The `coverage_percent` value should be a decimal number between 0 and 1.
- Each `file_path` value should start with 'src/'.
- All statements in the `source_coverage` dictionary should be integers.
- Missed statements should be less than or equal to missed ranges.
- The `aggreate_dir` attribute of the aggregator instance should not be None and point to a valid directory.
- The `aggregate()` method should return a non-None result.
- The `source_coverage` value in the result should be an instance of `SourceCoverageEntry`.
- The file path in the `SourceCoverageEntry` object should match the expected value.

COVERAGE

src/pytest_llm_report/aggregation.py	66 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 148-155, 157-159, 170, 182, 184-186, 194, 217, 219-220, 235, 245, 248-249, 251, 253, 276-279, 281)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test loading coverage from configured source file when option is not set.

Why Needed: This test prevents a potential bug where the aggregator fails to load coverage data when the `'llm_coverage_source'` option is not specified.

Key Assertions:

- Verify that `'_load_coverage_from_source()'` returns `'None'` when `'llm_coverage_source'` is `'None'`.
- Verify that `'_load_coverage_from_source()'` raises a `'UserWarning'` when `'llm_coverage_source'` is `'/nonexistent/coverage'`.
- Verify that `'_load_coverage_from_source()'` successfully loads coverage data from the mock `'coverage'` file.
- Verify that `'mock_cov.report()'` returns the correct coverage percentage (80.0) when called.
- Verify that `'mock_mapper.map_source_coverage()'` correctly maps source coverage to entries.
- Verify that `'mock_cov.load()'` is called once when `'_load_coverage_from_source()'` is run.
- Verify that `'mock_cov.report()'` was called during its execution.
- Verify that the correct entry is returned from `'_load_coverage_from_source()'` with a coverage percentage of 80.0.

COVERAGE

src/pytest_llm_report/aggregation.py	19 lines (ranges: 245-246, 248-249, 251, 253-257, 259, 262-263, 265-266, 269, 271-272, 274)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the aggregator recalculates the summary correctly when there are multiple test results.

Why Needed: To prevent regression in case of multiple failed tests, where the total duration is calculated incorrectly.

Key Assertions:

- The total number of tests passed should be equal to the original total.
- The number of failed tests should remain unchanged.
- The skipped tests should not affect the total count.
- The xfailed and xpassed counts should still be correct.
- The error count should also remain unchanged.
- The coverage percentage should be preserved.
- The total duration should match the original value.

COVERAGE

src/pytest_llm_report/aggregation.py	17 lines (ranges: 217, 219-233, 235)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test case verifies that skipping an invalid JSON file prevents a regression.

Why Needed: This test ensures that the aggregation function behaves correctly when handling invalid JSON files, preventing potential regressions.

Key Assertions:

- The test verifies that only valid reports are counted in the aggregate result.
- The test checks if the missing_fields.json file is skipped by the aggregator.
- The test asserts that a UserWarning is raised with the message 'Skipping invalid report file' when an invalid JSON file is encountered.
- The test ensures that the count of valid reports remains unchanged after skipping an invalid JSON file.

COVERAGE

src/pytest_llm_report/aggregation.py	71 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119-120, 125, 127-128, 148-153, 155, 157-159, 162, 164-166, 168, 170, 182, 184-186, 194, 217, 219-220, 235, 245, 248-249, 251, 253, 276-279, 281)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the aggregator recalculates the summary correctly when given a list of tests with coverage totals and a latest summary.

Why Needed: This test prevents regression where the aggregator fails to recalculate the summary after receiving new data, potentially causing incorrect results or misleading reports.

Key Assertions:

- The total number of tests passed in the summary should be equal to the number of tests provided.
- The total duration of all tests passed in the summary should be less than or equal to the latest summary's total duration.
- The coverage total percent of all tests passed in the summary should be greater than or equal to the latest summary's coverage total percent.
- At least one test should have failed in the summary, as indicated by a 'failed' key.
- All tests should have been covered at least once in the summary, as indicated by a 'passed' key.
- The total duration of all tests passed in the summary should be greater than or equal to the latest summary's total duration.

COVERAGE

src/pytest_llm_report/aggregation.py	10 lines (ranges: 44, 217, 219-225, 235)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The `cached_tests_are_skipped` test verifies that cached tests are skipped by the annotator.

Why Needed: This test prevents a regression where the annotator incorrectly skips cached tests.

Key Assertions:

- Mocking `mock_provider`, `mock_cache`, and `mock_assembler` to isolate dependencies.
- Verifying that the annotated function returns an empty list for cached tests.
- Checking that the annotated function does not raise any exceptions when called with mocked dependencies.
- Asserting that the annotated function correctly skips cached tests by checking if it returns a non-empty list.
- Verifying that the test passes only when no cached tests are available.
- Checking that the test fails when there are cached tests available to be skipped.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-81, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies concurrent annotation functionality in the test_anotate_tests module.

Why Needed: Prevents potential performance regressions or bugs that may arise from concurrent annotation requests.

Key Assertions:

- Ensures that multiple annotations are processed sequentially without any conflicts or delays.
- Verifies that cache invalidation is properly handled when annotations are updated concurrently.
- Checks for any synchronization issues that might occur due to concurrent access to the annotator's state.
- Demonstrates that the annotator can handle a large number of concurrent requests without significant performance degradation.
- Ensures that the annotator does not get stuck in an infinite loop when multiple annotations are processed concurrently.
- Verifies that cache invalidation is correctly propagated to subsequent annotation requests even after the initial request has completed.
- Checks for any potential issues related to thread safety or synchronization when accessing shared resources.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	64 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104-112, 129-135, 137, 139, 229-232, 234, 236-237, 239, 245-246, 248-253, 255, 261, 266)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The annotator handles failures concurrently when multiple tests are executed simultaneously.

Why Needed: This test prevents a potential issue where the annotator fails to handle concurrent failures, leading to inconsistent results or errors.

Key Assertions:

- Mocking the ``annotator`` function with multiple instances of ``mock_provider``, ``mock_cache``, and ``mock_assembler`` should not cause any failures.
- The ``capsys`` fixture should capture and display all output from the annotator tests.
- No exceptions should be raised when executing the test suite concurrently.
- All annotations should be generated correctly even if multiple tests fail.
- The ``annotator`` function should handle failures by logging errors or returning an error message.
- No inconsistent results or errors should be reported by the annotator.
- The annotator's output should not be affected by concurrent test execution.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104-112, 129-135, 137-139, 229-232, 234, 236-237, 239, 245-246, 248-253, 255, 261-264, 266)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The ``test_progress_reporting`` function is used to verify the progress reporting mechanism in the annotator.

Why Needed: This test prevents regressions that may occur when the progress reporting mechanism is not functioning correctly.

Key Assertions:

- Mocking ``mock_provider``, ``mock_cache``, and ``mock_assembler`` objects ensures they are properly initialized before use.
- Verifying that the progress bar updates correctly after each iteration of the test loop.
- Checking if the progress reporting messages are being displayed as expected on the console or output stream.
- Ensuring that the progress bar is not stuck at 100% for an extended period of time.
- Verifying that the progress reporting mechanism does not throw any exceptions when called with invalid arguments.
- Testing the progress reporting mechanism with different types of annotations (e.g., text, image, etc.) to ensure it works correctly for all cases.
- Checking if the progress bar is reset to 0% after each test iteration to maintain accurate tracking.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies that sequential annotation is performed correctly.

Why Needed: Prevents a potential bug where the annotator does not process annotations in sequence.

Key Assertions:

- mock_provider is called before mock_cache and before mock_assembler.
- mock_provider is called before any of its dependencies (mock_cache and mock_assembler).
- mock_cache is called after all of its dependencies (mock_provider, mock_assembler) have been called.
- mock_assembler is called only once, after all of its dependencies (mock_provider, mock_cache) have been called.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The ``test_skips_if_disabled`` test verifies that the annotator does not perform any action when the LLMS (Large Language Model Service) is disabled.

Why Needed: This test prevents a regression where the annotator might skip tests or annotations if the LLMS is disabled, potentially causing unintended consequences.

Key Assertions:

- The ``config`` object is created with ``provider='none'``, indicating that the provider is not enabled.
- No annotation is performed on any tests in the list ``[]``.
- The annotator does not attempt to annotate any tests.
- There are no test annotations generated or saved.
- The ``test_skips_if_disabled`` function returns without performing any action.
- The ``config`` object remains unchanged after calling ``annotate_tests()``.
- No error is raised when the LLMS is disabled, indicating that the annotator behaves as expected.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	2 lines (ranges: 45-46)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The annotator should skip the annotation process when a provider is unavailable.

Why Needed: This test prevents regression where an annotator might incorrectly annotate data due to a temporary network or provider issue.

Key Assertions:

- mock_provider.return_value.__class__.__name__ == 'MockProvider'
- mock_provider.return_value.is_available() == False
- self.assertEqual(mock_provider.return_value.annotations, [])
- mock_provider.return_value.skip_annotation()
- self.assertEqual(mock_provider.return_value.skip_annotation(), True)
- mock_provider.return_value._provider is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that annotator reports progress and first error when annotated concurrently with progress and errors.

Why Needed: To prevent regression in concurrent mode, where multiple annotations are performed simultaneously.

Key Assertions:

- Verify that the annotation process is reported correctly with both a success outcome and an error.
- Verify that the first error encountered during annotation is reported as expected.
- Check if the progress messages accurately reflect the number of tasks being processed.
- Ensure that LLM annotation messages are included in the progress messages.
- Verify that the annotated result matches the expected value (in this case, 2).
- Confirm that the first error message includes the relevant information about the task ID and outcome.
- Check if the progress messages include a clear indication of the number of tasks being processed.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	28 lines (ranges: 229-232, 234, 236-237, 239-242, 245-246, 248-253, 255-258, 261-264, 266)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Should wait if rate limit interval has not elapsed.

Why Needed: Prevents test failure due to incorrect sleep behavior when rate limit interval hasn't elapsed.

Key Assertions:

- The time.sleep function should be called with a delay of at least 0.1 seconds.
- The time.sleep function should be called with a delay of exactly 1 second.
- The time.sleep function should not be called with a delay less than 0.1 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	23 lines (ranges: 165-168, 170-171, 173-174, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the annotator reports progress when caching tests.

Why Needed: Prevents regression where cached tests are not reported with progress.

Key Assertions:

- The `progress_msgs` list should contain messages indicating cache status.
- Each message in `progress_msgs` should start with `(cache):`.
- At least one message in `progress_msgs` should be present.
- Any message in `progress_msgs` should have the format `test_cached`.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	37 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-84, 97-98, 100, 127, 129-135, 137, 139)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that when the provider is not available, it prints a message and returns without attempting to annotate tests.

Why Needed: This test prevents regression by ensuring that the annotator does not attempt to annotate tests when the provider is unavailable.

Key Assertions:

- `mocks.is_available` was called with `False`
- `annotate_tests` calls `mock_provider.get_provider` with `mock.Provider`
- `mock_provider.get_provider` returns a `MagicMock` instance
- `mock_provider.is_available` returns `False`
- `assert 'not available. Skipping annotations' in captured.out`

COVERAGE

<code>src/pytest_llm_report/collector.py</code>	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
<code>src/pytest_llm_report/llm/annotator.py</code>	7 lines (ranges: 45, 48-52, 54)
<code>src/pytest_llm_report/options.py</code>	3 lines (ranges: 107, 147, 224)
<code>src/pytest_llm_report/plugin.py</code>	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that extracting a malformed JSON from a response fails with an error message.

Why Needed: Prevents the test from passing if the extracted JSON is valid, allowing for coverage of invalid cases.

Key Assertions:

- The ``annotation`` variable should be set to ``jsonDecodeError``.
- The ``error`` attribute of ``annotation`` should contain a string 'Failed to parse LLM response as JSON'.
- The ``annotation.error`` attribute should have the correct type hint.
- The ``annotation.error`` attribute should not be ``None``.
- The ``annotation.error`` attribute should be an instance of ``str``.
- The ``annotation.error`` attribute should contain a string that starts with 'Failed to parse'.
- The ``annotation.error`` attribute should contain the string 'Failed to parse LLM response as JSON'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	10 lines (ranges: 52-53, 186-187, 190-191, 194-195, 220-221)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the `test_base_parse_response_non_string_fields` test verifies that non-string fields are handled correctly in the response data.

Why Needed: This test prevents a potential bug where the parser incorrectly handles non-string fields as lists, leading to incorrect results or errors.

Key Assertions:

- The correct scenario is set by the `scenario` key in the response data.
- The expected why needed value is correctly identified as 'list'.
- The correct key assertions are made using the `key_assertions` list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	22 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203-207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the `get_gemini_provider` function returns a `GeminiProvider` instance.

Why Needed: This test prevents regression in case the `Config` class is modified to use a different provider type (e.g., 'satellite') without updating the `get_provider` function.

Key Assertions:

- The returned value of `provider` should be an instance of `GeminiProvider`.
- The `provider` variable should hold a reference to a `GeminiProvider` instance.
- The `provider` attribute of the `config` object should have been set correctly using the `Config` class.
- The `get_provider` function should return a `GeminiProvider` instance when called with a valid configuration.
- If the `Config` class is modified to use a different provider type (e.g., 'satellite'), the test should fail and provide a clear error message.
- The `Config` class should be updated to include the correct provider type for the new provider.
- The `get_provider` function should be updated to handle cases where the provider is not found in the configuration.
- If an invalid provider type is passed to the `Config` constructor, the test should raise a meaningful error message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	10 lines (ranges: 52-53, 245, 247, 249, 252, 257, 262-263, 265)
src/pytest_llm_report/llm/gemini.py	7 lines (ranges: 134, 136-139, 141-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that a `ValueError` is raised when an unknown LLM provider is specified.

Why Needed: This test prevents a bug where the program incorrectly accepts an invalid LLM provider.

Key Assertions:

- The function ``get_provider()`` raises a ``ValueError`` with the message 'Unknown LLM provider: invalid'.
- The error message includes the string 'invalid' to identify the unknown provider.
- The test verifies that the ``pytest.raises()`` matcher is used correctly to catch the `ValueError`.
- The test checks that the ``match`` parameter of the ``pytest.raises()`` matcher matches the expected error message.
- The test ensures that the ``Config`` object passed to ``get_provider()`` has an invalid provider.
- The test verifies that the ``invalid`` value is used as the provider in the ``Config`` object.
- The test checks that the ``get_provider()`` function raises a ``ValueError`` with the specified error message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 245, 247, 249, 252, 257, 262, 267)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies that the `get_litellm_provider` function returns a correct instance of `LiteLLMProvider`.

Why Needed: Prevents a potential bug where the test fails due to an incorrect provider being returned.

Key Assertions:

- The `provider` attribute of the returned `LiteLLMProvider` object is set to `'litellm'`.
- The `provider` attribute of the returned `LiteLLMProvider` object is a string (`'litellm'`).
- The `provider` attribute of the returned `LiteLLMProvider` object is an instance of `LiteLLMProvider`.
- The `provider` attribute of the returned `LiteLLMProvider` object is set to `'litellm'` and has no additional attributes.
- The `provider` attribute of the returned `LiteLLMProvider` object is a string (`'litellm'`) with an empty string as its value.
- The `provider` attribute of the returned `LiteLLMProvider` object is not `None`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	9 lines (ranges: 52-53, 245, 247, 249, 252, 257-258, 260)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies that a NoopProvider is returned when the 'provider' parameter is set to 'none'

Why Needed: Prevents a potential bug where a valid provider is not found due to an incorrect or missing configuration.

Key Assertions:

- The function `get_provider` returns an instance of `NoopProvider` instead of another valid provider.
- The correct configuration for the 'provider' parameter is provided (in this case, 'none')
- A NoopProvider instance is created with the given configuration

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 245, 247, 249-250)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the `get_ollama_provider` method returns an instance of `OllamaProvider`.

Why Needed: Prevents a potential bug where an incorrect or malformed configuration is passed to the provider.

Key Assertions:

- The function `get_ollama_provider` in the `Config` class correctly creates an instance of `OllamaProvider` with the provided provider.
- The method `get_provider` in the `Config` class correctly uses the provided configuration to create a valid Ollama provider.
- The returned value from `get_ollama_provider` is indeed an instance of `OllamaProvider` as expected.
- A malformed or incorrect configuration would result in an error being raised instead of creating an invalid provider.
- The correct type hinting for the `provider` parameter ensures that only valid providers are accepted.
- The method name and signature match the expected behavior, indicating a successful implementation.
- No other assertions are necessary as this test verifies the basic functionality of the `get_ollama_provider` method.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 245, 247, 249, 252-253, 255)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies that the `is_available()` method returns `True` for a provider with no available caches.

Why Needed: This test prevents a regression where a provider without any available caches would incorrectly return `False` when checking availability.

Key Assertions:

- `provider.is_available()` is `True`
- `provider.checks == 1`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 107-108, 110-111)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the default model name is set to the configuration when a concrete provider is created.

Why Needed: This test prevents regression where the default model name is not set correctly in cases where a concrete provider is used.

Key Assertions:

- The ``get_model_name()`` method of the provider returns the value of ``model`` from the provided configuration.
- The ``model`` attribute of the provider instance has the same value as the ``model`` parameter passed to the ``Config`` constructor.
- The ``provider.get_model_name()`` call does not raise an exception if a concrete provider is used with a default model name.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 136)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the `get_rate_limits` method returns `None` when no rate limits are specified in the configuration.

Why Needed: This test prevents a potential bug where the default rate limits are not properly initialized or set, potentially leading to unexpected behavior.

Key Assertions:

- `config.get('rate_limit')` is `None`
- `provider.get_rate_limits() == None`
- `assert isinstance(provider.get_rate_limits(), types.NoneType)`
- `assert provider.get_rate_limits().get('default_rate_limit') is None`
- `assert provider.get_rate_limits().get('max_rate_limit') is None`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 128)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that `is_local()` returns False when the default is set to false.

Why Needed: Prevents regression where the default value of `is_local()` is incorrectly reported as True.

Key Assertions:

- The `provider.is_local()` method should return False when `is_local_defaults_to_false` is called with a config that sets `local_defaults_to_false` to false.
- The `provider.is_local()` method should not raise an exception when `is_local_defaults_to_false` is called with a config that sets `local_defaults_to_false` to true.
- The `provider.is_local()` method should correctly handle the case where `local_defaults_to_false` is set to false in the config.
- The `provider.is_local()` method should not report any errors or warnings when `is_local_defaults_to_false` is called with a config that sets `local_defaults_to_false` to true.
- The `provider.is_local()` method should correctly handle the case where `local_defaults_to_false` is set to false in the config and the provider is created with it.
- The `provider.is_local()` method should not report any errors or warnings when `is_local_defaults_to_false` is called with a config that sets `local_defaults_to_false` to true and the provider is created with it.
- The `provider.is_local()` method should correctly handle the case where `local_defaults_to_false` is set to false in the config and the provider is created with it, without raising an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 147)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The function `hash_source` is called with a source code string that produces the same hash value.

Why Needed: This test prevents a bug where different source codes produce different hashes, which could lead to unexpected behavior in caching or other applications.

Key Assertions:

- source_code_is_same
- hash_value_is_not_different
- source_code_hash_is_consistent_with_source_code_hash
- source_code_hash_is_consistent_with_hash_of_source_code
- source_code_hash_is_the_same_as_hash_of_source_code

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Testing the behavior of `hash_source` when different sources are used.

Why Needed: This test prevents a potential bug where two functions with the same source code but different names could produce the same hash value, leading to unexpected behavior in caching.

Key Assertions:

- The function `hash_source()` should return a different hash value for two different source strings.
- The function `hash_source()` should not be able to find a common prefix between two source strings.
- The function `hash_source()` should raise an error if the same source string is used multiple times.
- The function `hash_source()` should preserve the original order of source strings when comparing them for equality.
- The function `hash_source()` should not be able to cache a function with the same name as another function that uses different sources.
- A hash collision should occur between two different source strings and their corresponding cached functions.
- The function `hash_source()` should correctly handle source strings with multiple words or phrases separated by spaces.
- The function `hash_source()` should not cache a function if the same source string is used multiple times in the test suite.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify the length of the hash generated by HashSource.

Why Needed: Prevents a potential issue where the hash length is not consistent across different inputs.

Key Assertions:

- The hash should be exactly 16 characters long.
- The hash length should remain constant regardless of the input.
- The hash should not be shorter than 16 characters but also not longer than 15 characters.
- The hash should have a consistent character distribution across all possible inputs.
- The hash should not be affected by the order of the characters in the input string.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test clearing cache entries after adding some initial data.

Why Needed: Prevents regression in case the test is run multiple times with different input data.

Key Assertions:

- Verify that all cache entries are cleared after calling `clear()`.
- Ensure that no cached annotations are retrieved even after clearing the cache.
- Check if the cache directory remains as expected after clearing.

COVERAGE

src/pytest_llm_report/cache.py	26 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 129, 132-136, 141)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that annotations with errors are not cached.

Why Needed: Prevents regression in case of error annotations being cached.

Key Assertions:

- The annotation 'error' is present in the cache.
- The value associated with 'test::foo' is None after retrieval.
- The annotation type is correct ('error')
- The cache directory is set correctly using tmp_path / "cache".
- The error message is retrieved from the cache correctly.

COVERAGE

src/pytest_llm_report/cache.py	11 lines (ranges: 39-41, 53, 55-56, 86, 88, 118-119, 121)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test case 'get_missing' verifies that the function returns None for missing entries.

Why Needed: This test prevents a potential bug where the function does not return an error when trying to retrieve a non-existent cache entry.

Key Assertions:

- The function should return `None` when given a key that does not exist in the cache.
- The function should raise a `KeyError` exception with a meaningful message when given a key that does not exist in the cache.
- The function should check if the cache is empty before trying to retrieve a non-existent entry.
- The function should return an error message indicating that the cache is missing a required entry.
- The function should handle cases where the cache directory is not writable or has incorrect permissions.

COVERAGE

src/pytest_llm_report/cache.py	9 lines (ranges: 39-41, 53, 55-56, 118-119, 121)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that annotations are correctly stored and retrieved from the cache.

Why Needed: Prevents bypass by ensuring that annotations are persisted even after a test has completed.

Key Assertions:

- Verify that the annotation is set correctly in the cache.
- Check that the annotation can be successfully retrieved from the cache.
- Ensure that the retrieved annotation matches the expected scenario and confidence level.

COVERAGE

src/pytest_llm_report/cache.py	28 lines (ranges: 39-41, 53, 55, 58, 60-62, 68-73, 86, 90, 92, 94, 97-101, 103, 118-119, 121)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_collector.py::TestCollectorCollectionErrors::test_collection_error_structure

1ms  2

AI ASSESSMENT

Scenario: Test verifies that a collection error has the correct 'nodeid' and 'message' attributes.

Why Needed: Prevents a potential bug where a collection error is incorrectly structured, potentially leading to incorrect or missing information being reported.

Key Assertions:

- The 'nodeid' attribute of the CollectionError object should match the provided 'nodeid' value.
- The 'message' attribute of the CollectionError object should match the provided 'message' value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test verifies that the `get_collection_errors` method returns an empty list when the collection is initially empty.

Why Needed: This test prevents a potential regression where the `get_collection_errors` method may return incorrect results or raise an exception due to an empty collection.

Key Assertions:

- The `collector.get_collection_errors()` function should return an empty list.
- No exceptions should be raised when the collection is initially empty.
- All errors in the collection should be ignored.

COVERAGE

src/pytest_llm_report/collector.py	15 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 285)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Testing the default value of `llm_context_override` when it's not provided.

Why Needed: Prevents a potential bug where the default value of `llm_context_override` is set to `None`, potentially causing unexpected behavior in downstream code.

Key Assertions:

- The `llm_context_override` attribute is checked for being `None`.
- The `TestCaseResult` object has an `llm_context_override` attribute that matches the expected `None` value.
- The `nodeid` and `outcome` of the `TestCaseResult` match the expected values.
- The `result.llm_context_override` attribute is set to `None`, as expected.
- No exception is raised when calling `llm_context_override` on a `TestCaseResult` object.
- The `llm_context_override` attribute is not checked for being `None` in other test cases.
- The default value of `llm_context_override` is correctly set to `None`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_collector.py::TestCollectorMarkerExtraction::test_llm_opt_out_default_false

1ms



AI ASSESSMENT

Scenario: Test the default value of llm_opt_out for LLM Opt Out feature.

Why Needed: Prevents regression in case where llm_opt_out defaults to False without explicit opt-out.

Key Assertions:

- The llm_opt_out attribute is set to False.
- The TestCaseResult object has an llm_opt_out attribute that matches the expected value.
- The test passes if llm_opt_out is indeed False or if it's explicitly set to True.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_collector.py::TestCollectorOutputCapture::test_capture_disabled_by_default

1ms



AI ASSESSMENT

Scenario: The test verifies that the output capture feature is disabled by default.

Why Needed: This test prevents a regression where the output capture feature was enabled by default.

Key Assertions:

- config.capture_failed_output should be set to False
- output_capture_enabled should not be True
- capture_mode should be 'disabled' or None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The 'TestCollectorOutputCapture' test verifies that the default value of 'capture_output_max_chars' is 4000.

Why Needed: This test prevents a potential issue where the default max chars value is not set correctly, potentially leading to unexpected behavior or errors in the application.

Key Assertions:

- The 'capture_output_max_chars' configuration option should be set to 4000 by default.
- The current value of 'capture_output_max_chars' should match 4000.
- If the default max chars is not set, the test will fail with an error message indicating that it's not a valid value.
- The application may throw an exception or behave unexpectedly if the default max chars is not set correctly.
- The test ensures that the 'capture_output_max_chars' configuration option is properly initialized and configured.
- If the default max chars is not set, the test will fail with a clear and descriptive error message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_collector.py::TestCollectorXfailHandling::test_xfail_failed_is_xfailed

1ms  3

AI ASSESSMENT

Scenario: Test 'xfail failures should be recorded as xfailed' verifies that failed test cases are correctly marked as xfailed in the report.

Why Needed: This test prevents regression where a failed test case is incorrectly marked as passed instead of xfailed.

Key Assertions:

- The `results` dictionary contains an entry for the specified nodeid with a value of 'xfailed'.
- The `outcome` attribute of the result is set to 'xfailed'.
- The `wasxfail` attribute of the report is set to 'expected failure'.

COVERAGE

src/pytest_llm_report/collector.py	36 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-118, 124, 127, 140, 155-159, 163, 167, 171, 209-210, 212, 216, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that when an xfail is passed, it should be recorded as xpassed in the report.

Why Needed: This test prevents regression where an xfail is not properly handled and instead records it as failed.

Key Assertions:

- the ``results`` dictionary contains a key with the value 'xpassed' for the nodeid `'test_xfail.py::test_unexpected_pass'``.
- the ``outcome`` attribute of the result is set to 'xpassed'.
- the ``wasxfail`` attribute of the report is set to 'expected failure'.

COVERAGE

src/pytest_llm_report/collector.py	26 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-115, 124, 127, 140, 155-159, 163, 167, 171, 209-210, 212-214)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the `create_collector` method of `TestCollector` class.

Why Needed: The test ensures that a new `TestCollector` instance is created with an empty collection and no errors.

Key Assertions:

- The `results` attribute of the collector should be an empty dictionary.
- The `collection_errors` list should be an empty list.
- The `collected_count` attribute should be set to 0.
- A new instance of `TestCollector` should be created with a Config object.
- No errors should be reported by the collector during initialization.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the ``get_results`` method returns a sorted list of node IDs from the collected results.

Why Needed: This test prevents a regression where the order of node IDs in the results is not guaranteed to be consistent across different runs, potentially leading to incorrect analysis or reporting.

Key Assertions:

- The ``nodeid`` attribute of each result object contains the correct value.
- The list of node IDs returned by the ``get_results`` method is sorted in ascending order.
- No duplicate node IDs are present in the sorted list.
- All nodes with a 'passed' outcome are included in the sorted list.
- No nodes without an outcome ('failed') are included in the sorted list.
- The sorting is stable, meaning that if two results have the same ``nodeid``, their original order is preserved.
- No duplicate node IDs are present in the sorted list of node IDs.

COVERAGE

src/pytest_llm_report/collector.py	15 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the `handle_collection_finish` method to ensure it correctly tracks collected and deselected counts.

Why Needed: This test prevents a potential bug where the count of collected items is not updated correctly after the collection finish.

Key Assertions:

- The `collected_count` attribute should be set to 3 (the number of collected items).
- The `deselected_count` attribute should be set to 1 (the number of deselected items).

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 78-79, 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the collector does not capture output when config is disabled and handle_report is used for integration via handle_runtest_logreport.

Why Needed: To prevent capturing of output in scenarios where the `capture_failed_output` configuration is set to False, allowing for integration with handle_report for reporting purposes.

Key Assertions:

- The collector does not capture any output when run on a test that was previously failed and has its `capture_failed_output` config set to False.
- The collector's results do not contain any captured stdout data.
- No error message is emitted by the collector due to no captured stdout.
- The collector's `results` dictionary does not contain a key named 't' or any other relevant keys.
- The collector's `results` dictionary contains an empty string value for the 'output' key.
- The collector's `results` dictionary contains False values for all other keys (passed, failed, skipped).
- The collector's `results` dictionary does not contain a 't' key with a non-empty string value.

COVERAGE

src/pytest_llm_report/collector.py	36 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-118, 124, 127-128, 130, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `capture_output` method captures stderr correctly.

Why Needed: This test prevents a potential bug where the `capture_output` method does not capture stderr.

Key Assertions:

- The `captured_stderr` attribute of the `TestCaseResult` object should be set to 'Some error'.
- The `report.capstderr` attribute should contain the string 'Some error'.
- The `report.capstdout` attribute should be an empty string.
- The `collector.capture_output(result, report)` method should call `result.captured_stderr = 'Some error'`.
- The `collector.capture_output(result, report)` method should set `captured_stderr` to the captured stderr value.
- The `report.capstderr` attribute should be updated with the captured stderr value.
- The `collector.capture_output(result, report)` method should update the `report` object correctly.
- The `collector.capture_output(result, report)` method should not raise any exceptions.
- The `report` object should have a non-empty `capstderr` attribute after calling `_capture_output`.

COVERAGE

src/pytest_llm_report/collector.py	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264, 268-269)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `capture_output` method captures stdout correctly.

Why Needed: This test prevents a potential bug where the captured stdout is not properly recorded.

Key Assertions:

- The `captured_stdout` attribute of the `TestCaseResult` object should contain the expected output.
- The `report.capstdout` attribute should set the correct value for stdout.
- The `collector.capture_output(result, report)` method should record the captured stdout correctly.
- The `result.captured_stdout` attribute should be equal to the captured stdout.
- The `report.capstderr` attribute is not used in this test and can be safely ignored.
- The `collector.capture_output(result, report)` method does not modify the original output.
- The `collector.capture_output(result, report)` method does not record any additional information.

COVERAGE

src/pytest_llm_report/collector.py	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264-265, 268)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `test_capture_output_truncated` function truncates output exceeding max chars.

Why Needed: This test prevents a potential bug where the collector fails to truncate output exceeding the maximum characters set in the configuration.

Key Assertions:

- The captured stdout should be truncated to '1234567890' if it exceeds the specified max_chars.
- The `captured_stdout` attribute of the `TestCaseResult` object should contain only the truncated output.
- The `report.capstdout` attribute should not exceed the maximum characters set in the configuration.
- The `report.capstderr` attribute is not used in this test and can be ignored for this test.
- The `collector._capture_output` method should call the `report.capstdout` method to update the captured stdout.
- The `result.captured_stdout` attribute should contain only the truncated output after calling `_capture_output`.
- The `report.capstderr` attribute should not be affected by the truncation of `captured_stdout`.

COVERAGE

src/pytest_llm_report/collector.py	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264-265, 268)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test creates a result with item markers.

Why Needed: This test prevents regression where the collector does not extract item markers correctly, leading to incorrect results.

Key Assertions:

- `item.get_closest_marker('llm_opt_out')` returns `MagicMock()`.
- `item.get_closest_marker('llm_context')` returns `MagicMock()`.
- `item.get_closest_marker('requirement')` returns `MagicMock()`.
- `result.param_id` is set to `'param1'`.
- `result.llm_opt_out` is `True`.
- `result.llm_context_override` is set to `'complete'`.
- `result.requirements` contains `['REQ-1', 'REQ-2']`.

COVERAGE

<code>src/pytest_llm_report/collector.py</code>	35 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 155-159, 163-164, 167-169, 171, 181-182, 185-189, 198-200, 209-210)
<code>src/pytest_llm_report/options.py</code>	2 lines (ranges: 107, 147)
<code>src/pytest_llm_report/plugin.py</code>	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test should handle RePrFileLocation causing crash report.

Why Needed: This test prevents a potential crash when RePrFileLocation is used in the error representation.

Key Assertions:

- The `report.longrepr` attribute is set to 'Crash report'.
- The `report.longrepr.__str__.return_value` is set to 'Crash report'.
- The `collector._extract_error(report)` function returns 'Crash report' as expected.

COVERAGE

src/pytest_llm_report/collector.py	22 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `_extract_error` method returns the correct string for a maximal error message.

Why Needed: This test prevents a potential regression where the `longrepr` attribute is not correctly propagated to the extracted error string.

Key Assertions:

- The value of `report.longrepr` should be equal to 'Some error occurred'.
- The method `_extract_error` should return the correct string for a maximal error message.

COVERAGE

src/pytest_llm_report/collector.py	22 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `_extract_skip_reason` method returns `None` when no `longrepr` is provided.

Why Needed: Prevents a potential bug where the method does not handle cases with no `longrepr` correctly.

Key Assertions:

- The `_extract_skip_reason` method should return `None` for an empty or missing `longrepr` attribute.
- The `_extract_skip_reason` method should not raise any exceptions when `longrepr` is `None`.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250, 252)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_collector_maximal.py::TestCollectorInternals::test_extract_skip_reason_string

1ms  3

AI ASSESSMENT

Scenario: Test `test_extract_skip_reason_string` verifies that the `_extract_skip_reason` method returns a string when given a `report` object.

Why Needed: Prevents regression in case of unexpected report longrepr values, which could lead to incorrect skip reasons being returned.

Key Assertions:

- The `report.longrepr` attribute is set to 'Just skipped'.
- The `_extract_skip_reason` method returns the expected string value.
- No other assertions are performed by this test.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250-251)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that extract skip reason tuple is called correctly.

Why Needed: This test prevents a potential bug where the `extract_skip_reason` method does not handle tuples with more than two elements.

Key Assertions:

- The `longrepr` attribute of the report object contains the expected file, line and message.
- The `longrepr` attribute of the report object is a tuple containing the specified file, line and message.
- When a tuple with three elements is passed to `_extract_skip_reason`, it correctly extracts the skip reason from the tuple.
- When a tuple with more than two elements is passed to `_extract_skip_reason`, it raises an `AssertionError` with a meaningful error message.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250-251)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: When the `handle_collection_report` method is called with a report that indicates a collection error, then it should record this error in the `collection_errors` list.

Why Needed: This test prevents a potential regression where the collector might not handle collection errors correctly and instead silently ignore them.

Key Assertions:

- The `collection_errors` list should contain exactly one item with `nodeid = 'test_broken.py'` and `message = 'SyntaxError'`.
- The error message in the first `collection_errors` item should be `'SyntaxError'`.
- All other items in the `collection_errors` list should have a different `nodeid` and/or an empty `message` field.

COVERAGE

src/pytest_llm_report/collector.py	21 lines (ranges: 58, 60-65, 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test 'handle_runtest_rerun' verifies that the `rerun` attribute of a report is correctly set to 1 after rerunning the test.

Why Needed: This test prevents regression in handling reruns, ensuring that reports with a `rerun` attribute are updated correctly when the test is rerun.

Key Assertions:

- res.rerun_count should be equal to 1
- res.final_outcome should be 'failed'
- report.wasxfail should not be present in report

COVERAGE

src/pytest_llm_report/collector.py	36 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-118, 124, 127-128, 130, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test Collector should handle run test setup failure correctly.

Why Needed: This test prevents a regression where the collector fails to record setup errors, potentially leading to incorrect reporting of test failures.

Key Assertions:

- res.outcome is set to 'error' as expected.
- res.phase is set to 'setup' as expected.
- res.error_message is set to 'Setup failed' as expected.

COVERAGE

src/pytest_llm_report/collector.py	36 lines (ranges: 90, 93-94, 96, 99-103, 109-112, 114-115, 124, 127, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test case: Handle runtest teardown failure

Why Needed: Prevents regression in case of teardown failure after a pass.

Key Assertions:

- The `teardown` report is not recorded as an error.
- The `teardown` report has the correct phase ('teardown') and error message ('Cleanup failed').
- The `results` dictionary contains the expected outcome ('error'), phase, and error message for test 't::f'.

COVERAGE

src/pytest_llm_report/collector.py	38 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-115, 124, 127-128, 130, 132-133, 135-137, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the GeminiProvider's `_parse_preferred_models` method with edge cases, specifically when no models are provided.

Why Needed: This test prevents a bug where the GeminiProvider does not correctly handle scenarios where no models are specified in the configuration.

Key Assertions:

- The 'm1' and 'm2' models should be present in the parsed list of preferred models.
- The 'All' model should also be present in the parsed list of preferred models if it is specified in the configuration.
- An empty list of preferred models should be returned when no models are provided in the configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	17 lines (ranges: 134, 136-139, 141-142, 385, 387, 417-424)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the rate limiter does not allow excessive tokens when there are available slots but too many requests.

Why Needed: This test prevents a potential bug where the rate limiter allows an edge case (excessive tokens) while still allowing enough available slots for other requests.

Key Assertions:

- assert limiter.next_available_in(60) > 0
- assert limiter.next_available_in(10) == 0
- assert limiter.record_tokens(50) < 100

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	35 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 81-82, 84, 87-88, 92-93, 95-96, 100-101, 103, 105, 107-114)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the `models_to_dict` method returns accurate coverage percentages for `SourceCoverageEntry` and `LlmAnnotation` objects.

Why Needed: The test prevents regression in coverage reporting when using models to dict variants, as it ensures that the coverage percentage is always reported correctly even with errors or timeouts.

Key Assertions:

- `d['coverage_percent'] == 50.0`
- `ann.to_dict()['error'] == 'timeout'`
- `meta.to_dict()['duration'] == 1.0`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	46 lines (ranges: 71-78, 104-107, 109, 111-113, 115, 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Tests the `CoverageMapper` class to ensure it correctly initializes with a given configuration.

Why Needed: Prevents potential bugs or regressions where a `CoverageMapper` instance is created without a valid configuration.

Key Assertions:

- The `config` attribute of the `CoverageMapper` instance should be set to the provided `Config` object.
- The `warnings` attribute of the `CoverageMapper` instance should be an empty list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	2 lines (ranges: 44-45)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies the `get_warnings` method returns a list of warnings as expected.

Why Needed: Prevents test failures due to incorrect handling of warnings in coverage reports.

Key Assertions:

- The `get_warnings` method is called on an instance of `CoverageMapper` with a valid configuration.
- The returned value is checked to be an instance of `list` as expected.
- A warning is extracted from the coverage report and added to the list of warnings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	3 lines (ranges: 44-45, 308)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `map_coverage` method returns an empty dictionary when no coverage file is found.

Why Needed: Prevents a potential bug where the test fails due to missing coverage data.

Key Assertions:

- The `mapper.map_coverage()` method should return an empty dictionary when `Path.exists` and `glob.glob` return False.
- The `mapper.warnings` list should contain at least one warning message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `CoverageMapper` extracts node IDs for all phases when `include_phase=all`.

Why Needed: This test prevents a regression where the coverage map does not include all phases when `include_phase=all`.

Key Assertions:

- The `_extract_nodeid()` method returns the expected node ID for each phase.
- The `_extract_nodeid()` method includes all phases in the coverage map.
- The `_extract_nodeid()` method excludes only the 'setup' phase from the coverage map when `include_phase=all`.
- The `CoverageMapper` class correctly handles the `include_phase=all` parameter.
- The test passes without any errors or warnings for this specific scenario.
- The test covers all possible cases where `include_phase=all` is used.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	11 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the `extract_nodeid` method with an empty context.

Why Needed: Prevents a potential bug where the method returns `None` for an empty context, potentially causing unexpected behavior or errors in downstream code.

Key Assertions:

- The `_extract_nodeid` method should return `None` when passed an empty string.
- The `_extract_nodeid` method should return `None` when passed `None` as the context.
- The method should not throw any exceptions or raise errors for these inputs.
- The method's behavior should be consistent with its documentation and other tests.
- The test should verify that the method returns `None` in both cases, without attempting to extract a node ID from an empty string or None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	4 lines (ranges: 44-45, 216-217)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the `test_extract_nodeid_filters_setup` test case filters out setup phase when `include_phase=run`.

Why Needed: This test prevents a bug where the coverage map includes nodes from the setup phase even though it's excluded.

Key Assertions:

- The `_extract_nodeid` method of the `CoverageMapper` class returns `None` for the given nodeid.
- The `include_phase` parameter is set to `'run'` in the test configuration.
- The coverage map does not include nodes from the setup phase when `include_phase=run`.
- The `test_foo` function is part of a module that has a setup phase, but it's excluded by default.
- The `test_foo` function should be filtered out from the coverage report.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	9 lines (ranges: 44-45, 216, 220, 224-225, 228-230)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the `extract_nodeid` method extracts the correct `nodeid` from the run phase context.

Why Needed: This test prevents a potential bug where the extracted `nodeid` is incorrect due to missing or incomplete information in the run phase context.

Key Assertions:

- The `_extract_nodeid` method of the `CoverageMapper` class correctly extracts the `nodeid` from the provided string.
- The `nodeid` extracted from the run phase context matches the expected value (`test.py::test_foo`).
- The test does not fail when the input string is empty or contains only whitespace characters.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	11 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test extracts contexts for full logic coverage of `_extract_contexts` method.

Why Needed: Prevents regression in coverage analysis when the `_extract_contexts` method is called with a file that has multiple test files but no other code.

Key Assertions:

- `assert 'test_app.py::test_one' in result`
- `assert 'test_app.py::test_two' in result`
- `assert len(one_cov) == 1 and one_cov[0].line_count == 2`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	57 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 137-140, 144, 148, 150, 152-153, 156, 160-163, 165, 167-168, 173, 176, 178-184, 187-189, 191-194, 196, 199-200, 202, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	17 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64, 100, 103, 111-112, 116, 123)
src/pytest_llm_report/util/ranges.py	13 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65-67)

AI ASSESSMENT

Scenario: Test 'test_extract_contexts_no_contexts' verifies that the function correctly handles data with no test contexts by returning an empty dictionary.

Why Needed: This test prevents a regression where the function incorrectly returns a non-empty dictionary for data without test contexts.

Key Assertions:

- mock_data.measured_files.return_value == ['app.py']
- mock_data.contexts_by_lineno.return_value == {}
- result == {}

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	14 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 144-146)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test extracts node ID variants for setup and teardown phases.

Why Needed: Prevents regression in coverage analysis by ensuring that all nodes are covered during both setup and teardown phases.

Key Assertions:

- The function `_extract_nodeid` returns the expected node ID for each test file.
- The function `_extract_nodeid` filters out nodes that do not belong to the specified phase (setup or teardown).
- The function `_extract_nodeid` does not return any nodes when there are no matching phases (e.g., 'test.py::test_no_phase').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	14 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231-234, 236, 239)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `test_load_coverage_data_no_files` function correctly handles the case when no coverage files exist.

Why Needed: This test prevents a potential bug where the `CoverageMapper` class does not handle the case when there are no coverage files.

Key Assertions:

- The function should return `None` and have exactly one warning.
- The first warning should be for code 'W001'.
- No other warnings should be present in the result.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	9 lines (ranges: 44-45, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test Load Coverage Data Read Error: Tests the function `_load_coverage_data()` when it encounters an error while reading a coverage file.

Why Needed: Prevents a potential bug where the test fails due to unexpected errors in coverage data loading, ensuring the function remains reliable and handles edge cases correctly.

Key Assertions:

- The function `_load_coverage_data()` should return `None` when it encounters an error while reading a coverage file.
- Any warnings generated by the function should contain the message 'Failed to read coverage data'.
- The function should not raise any exceptions during execution, maintaining its robustness and reliability.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	17 lines (ranges: 44-45, 72-73, 83, 86, 88, 92, 94-96, 107-111, 114)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test should handle parallel coverage files from xdist and verify that the CoverageMapper correctly updates its internal state.

Why Needed: This test prevents regression where the CoverageMapper fails to update its internal state when loading coverage data with parallel files from xdist, potentially leading to incorrect or missing coverage information.

Key Assertions:

- assert mock_main_data.update.call_count >= 2
- assert mock_parallel_data1.call_count == 0
- assert mock_parallel_data2.call_count == 0

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	15 lines (ranges: 44-45, 72-73, 83, 86, 88, 92, 94, 98, 101-104, 106)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the ``map_coverage`` method when it does not receive any coverage data.

Why Needed: Prevents a potential bug where the test fails due to an incorrect assumption about the behavior of ``_load_coverage_data`` when no data is available.

Key Assertions:

- The function should return an empty dictionary ``{}`` when ``_load_coverage_data`` returns `None`.
- No exception should be raised when ``_load_coverage_data`` returns `None`.
- The test should pass even if the ``_load_coverage_data`` method returns a non-`None` value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	5 lines (ranges: 44-45, 58-60)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the ``map_source_coverage`` method skips files with errors during analysis.

Why Needed: This test prevents a regression where an error in the `analysis2` function would cause all source code to be skipped.

Key Assertions:

- `mock_data.measured_files.return_value` should return `['app.py']`
- `mock_cov.get_data.return_value` should raise `Exception('Analysis failed')`
- entries should not contain any files with errors

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	22 lines (ranges: 44-45, 243-244, 246-248, 250, 252-254, 259, 261, 263-268, 271, 299-300)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116, 123)

AI ASSESSMENT

Scenario: Test 'Should exercise all paths in map_source_coverage' to ensure comprehensive coverage of source files.

Why Needed: This test prevents regression by ensuring that the CoverageMapperMaximal class exercises all possible paths in the map_source_coverage configuration.

Key Assertions:

- The function `map_source_coverage` should return a list containing exactly one entry with the following properties: `file_path`, `statements`, `covered`, `missed`, and `coverage_percent`.
- The value of `file_path` in the returned entry should be 'app.py'.
- The number of statements in the returned entry should be 3.
- The value of `covered` in the returned entry should be 2.
- The number of missed files in the returned entry should be 1.
- The percentage of covered lines in the returned entry should be 66.67 (rounded to two decimal places).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	32 lines (ranges: 44-45, 243-244, 246-248, 250, 252, 259-261, 273, 276-279, 281-283, 285-293, 295, 299-300)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	17 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64, 100, 103, 111-112, 116, 123)
src/pytest_llm_report/util/ranges.py	14 lines (ranges: 29, 33, 35-37, 39-40, 45-47, 50, 52, 65-66)

AI ASSESSMENT

Scenario: Test the `make_warning` factory function to ensure it correctly identifies and handles unknown warnings.

Why Needed: This test prevents a potential bug where an unknown warning is incorrectly classified as having no coverage.

Key Assertions:

- The `make_warning` factory function should return a `Warning` object with the correct code (`W001_NO_COVERAGE`) and message.
- The `message` attribute of the returned `Warning` object should contain the expected string `'Unknown warning.'`
- The `detail` attribute of the returned `Warning` object should be set to the specified value `'test-detail'`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that warning codes have correct values.

Why Needed: This test prevents a potential regression where the warning code values are incorrect, which could lead to unexpected behavior or errors in downstream code.

Key Assertions:

- {'message': 'assert WarningCode.W001_NO_COVERAGE.value == "W001"', 'expected_value': 'W001'}
- {'message': 'assert WarningCode.W101_LLM_ENABLED.value == "W101"', 'expected_value': 'W101'}
- {'message': 'assert WarningCode.W201_OUTPUT_PATH_INVALID.value == "W201"', 'expected_value': 'W201'}
- {'message': 'assert WarningCode.W301_INVALID_CONFIG.value == "W301"', 'expected_value': 'W301'}
- {'message': 'assert WarningCode.W401_AGGREGATE_DIR_MISSING.value == "W401"', 'expected_value': 'W401'}

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the `Warning.to_dict()` method to ensure it returns a dictionary with correct keys.

Why Needed: This test prevents a potential bug where the `Warning.to_dict()` method does not return a dictionary with all required keys.

Key Assertions:

- The function ``to_dict()`` should return a dictionary with keys `'code'`, `'message'`, and `'detail'`.
- The value of `'code'` should be set to the correct warning code.
- The value of `'message'` should be set to the correct warning message.
- The value of `'detail'` should be set to the correct detail message if it exists.
- If a detail message is present, its length should not exceed 50 characters.
- If no detail message is present, the `'detail'` key should be empty.
- The function should raise an `AssertionError` with a meaningful error message if any of the assertions fail.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	6 lines (ranges: 70-72, 74-76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_errors_maximal.py::TestMakeWarning::test_make_warning_kno
wn_code

1ms  3

AI ASSESSMENT

Scenario: Test verifies that a warning with the standard message is created when known code is used.

Why Needed: This test prevents a potential regression where warnings are not correctly generated for known code.

Key Assertions:

- `w.code == WarningCode.W101_LLM_ENABLED`
- `w.message == WARNING_MESSAGES[WarningCode.W101_LLM_ENABLED]`
- `w.detail` is `None`

COVERAGE

<code>src/pytest_llm_report/collector.py</code>	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
<code>src/pytest_llm_report/errors.py</code>	4 lines (ranges: 139-142)
<code>src/pytest_llm_report/plugin.py</code>	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test MakeWarning::test_make_warning_unknown_code verifies that the test uses a fallback message for unknown code.

Why Needed: This test prevents a regression where the typed function would not provide a warning when given an unknown WarningCode.

Key Assertions:

- The test should restore the original message after restoring the missing code.
- The test should assert that the restored message is 'Unknown warning.'
- The test should not assert any other messages or values than 'Unknown warning.'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test 'test_make_warning_with_detail' verifies that a warning is created with the correct code and detail.

Why Needed: This test prevents a potential regression where a warning might not be created correctly due to an invalid configuration.

Key Assertions:

- The function `make_warning` returns a Warning object with the correct `code` attribute set to `WarningCode.W301_INVALID_CONFIG` and the correct `detail` attribute set to 'Bad value'.
- The function `make_warning` returns a Warning object with the correct `code` attribute set to `WarningCode.W301_INVALID_CONFIG`.
- The function `make_warning` returns a Warning object with the correct `detail` attribute set to 'Bad value'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Ensures that enum values are indeed strings and start with 'W' to prevent Warnings.

Why Needed: This test prevents a potential warning when trying to use non-string enum values.

Key Assertions:

- `assert isinstance(code.value, str)`
- `assert code.value.startswith('W')`
- `code.value` should be a string
- `code.value` should start with 'W'
- `WarningCode.values()` should return only strings
- `WarningCode.values()` should include values starting with 'W'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_errors_maximal.py::TestWarningDataClass::test_warning_to_dict_no_detail

1ms



AI ASSESSMENT

Scenario: The test verifies that the Warning class can be serialized into a dictionary without including detailed information.

Why Needed: This test prevents a potential bug where the warning details are included in the serialization of the Warning object to a dictionary.

Key Assertions:

- The 'code' key is present with value 'W001'
- The 'message' key is present with value 'No coverage'
- The 'code' and 'message' keys have the correct values
- The warning details are not included in the dictionary
- The dictionary has the expected structure

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	5 lines (ranges: 70-72, 74, 76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_errors_maximal.py::TestWarningDataClass::test_warning_to_dict_with_detail

1ms



AI ASSESSMENT

Scenario: Test the warning to dictionary conversion with detailed information.

Why Needed: This test prevents a potential bug where warnings are not properly serialized in dictionaries.

Key Assertions:

- The 'code' key should be present and have the correct value ('W001')
- The 'message' key should be present and have the correct value ('No coverage')
- The 'detail' key should be present and have the correct value ('Check setup')

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	6 lines (ranges: 70-72, 74-76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies that the ``is_python_file`` function returns `False` for non-.py files.

Why Needed: Prevents a potential bug where the function incorrectly identifies Python files as non-Python files, potentially leading to incorrect file classification and implications in downstream code.

Key Assertions:

- The function should return ``False`` when given a non-.py file path (e.g., ``foo/bar.txt``).
- The function should not return ``False`` for ``.pyc`` files (e.g., ``foo/bar.pyc``).
- When the input is a valid Python file, the function should correctly identify it as a Python file.
- If an invalid path is passed to the function, it should raise an error or handle it in a way that makes sense for the application.
- The function should not have any side effects (e.g., modifying external state) when determining whether a file is Python or not.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	1 lines (ranges: 79)

AI ASSESSMENT

Scenario: Verifies that the ``is_python_file`` function returns `True` for a Python file.

Why Needed: Prevents a potential bug where the function incorrectly identifies non-Python files as such.

Key Assertions:

- The function should correctly identify ``.py`` files and return ``True``.
- The function should not incorrectly identify other types of files (e.g. ``.txt``, ``.js``) as Python files.
- The function should handle file paths with leading or trailing whitespace correctly.
- The function should ignore case when comparing file extensions (e.g. ``.Py`` vs ``.py``).
- The function should raise an error if the input is not a string or a valid file path.
- The function should be able to handle files with relative paths correctly (e.g. ``.foo/bar.py``).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	1 lines (ranges: 79)

AI ASSESSMENT

Scenario: Verifies that the ``make_relative`` function correctly makes a path relative to the test directory.

Why Needed: This test prevents a potential bug where the function does not handle cases where the input file path is absolute.

Key Assertions:

- The function should be able to create an intermediate directory if it doesn't exist and then move the file into that directory.
- The function should remove any existing intermediate directory before creating it.
- The function should preserve the original file name and extension.
- The function should handle cases where the input file path is absolute (e.g., ``/path/to/file.py``)
- The function should not create an intermediate directory if the input file path is already relative (e.g., ``./file.py``)
- The function should preserve the original file permissions and ownership.
- The function should handle cases where the test directory does not exist (i.e., ``tmp_path`` is `None`)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64)

AI ASSESSMENT

Scenario: The test verifies that the `make_relative` function returns a normalized path when there is no base.

Why Needed: This test prevents potential issues where an invalid or empty base directory causes unexpected behavior in the application.

Key Assertions:

- `result == 'foo/bar'`
- is not equal to 'foo' (should be normalized)
- is not equal to 'bar' (should be normalized)
- is not equal to 'foo/bar' (should be normalized)
- has a length of 3 (expected to have 2 parts: 'foo', '.', and '/')
- does not contain any leading slashes (expected to be normalized)
- does not contain any trailing slashes (expected to be normalized)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	7 lines (ranges: 30, 33, 36, 39, 42, 55-56)

AI ASSESSMENT

Scenario: The test verifies that a normalized path is returned for an already-normalized input.

Why Needed: This test prevents a potential bug where the `normalize_path` function would incorrectly return the original input if it's already normalized.

Key Assertions:

- The normalized path should be the same as the original input.
- The function should not modify the input path.
- The function should handle paths with leading or trailing slashes correctly.
- The function should ignore any redundant separators (e.g., multiple dots in a file name).
- The function should preserve the directory structure of the input path.
- The function should raise an error if the input is not a string or a Path object.
- The function should handle paths with non-ASCII characters correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

AI ASSESSMENT

Scenario: Tests the `normalize_path` function for forward slashes.

Why Needed: Prevents a bug where the function incorrectly converts forward slashes to backslashes in certain paths.

Key Assertions:

- The function should correctly convert `'foo\bar'` to `'foo/bar'`.
- The function should not convert `'\bar'` to `'/bar'`.
- The function should handle multiple consecutive backslashes correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

AI ASSESSMENT

Scenario: Verifies that the `normalize_path` function strips trailing slashes from paths.

Why Needed: Prevents a potential bug where a path with a trailing slash is returned as is, potentially causing issues downstream.

Key Assertions:

- The input path does not end with a forward slash (`'/'`).
- The output path has no leading forward slashes (`'/'`).
- The function correctly handles paths with multiple levels of nesting (e.g., `'foo/bar/baz'`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

AI ASSESSMENT

Scenario: Verifies whether the `should_skip_path` function correctly skips paths matching custom patterns.

Why Needed: This test prevents a potential bug where the function does not skip paths that should be excluded due to custom patterns.

Key Assertions:

- The function `should_skip_path` is called with a path `'tests/conftest.py'` and an exclude pattern `['test*']`
- The function `should_skip_path` returns `True` for the path `'src/module.py'`
- The function `should_skip_path` returns `False` for the path `'tests/conftest.py'`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	15 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116-117, 119-121, 123)

AI ASSESSMENT

Scenario: tests/test_fs.py::TestShouldSkipPath::test_normal_path**Why Needed:** To ensure that the 'should_skip_path' function correctly handles normal file system paths.**Key Assertions:**

- The function should return True for a normal path (e.g. 'src/module.py').
- The function should not return False for a normal path (e.g. 'src/module.py').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116, 123)

AI ASSESSMENT

Scenario: The test verifies that the 'should_skip_path' function correctly identifies '.git' directories.**Why Needed:** This test prevents a potential bug where the function incorrectly skips non-'.git' directories.**Key Assertions:**

- assert should_skip_path('.git/objects/foo') is True
- assert not should_skip_path('non_git_directory.txt')

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

AI ASSESSMENT

Scenario: The test verifies that the ``should_skip_path`` function correctly skips the ``__pycache__`` directory.

Why Needed: This test prevents a regression where the ``should_skip_path`` function does not skip the ``__pycache__`` directory, causing unexpected behavior in tests that rely on it.

Key Assertions:

- The path should be skipped by the ``should_skip_path`` function.
- The ``__pycache__`` directory is skipped by the ``should_skip_path`` function.
- The ``should_skip_path`` function returns `True` for paths within the ``__pycache__`` directory.
- The test case asserts that the path is not included in the cache.
- The ``should_skip_path`` function should be able to determine whether a path is cached or not.
- The ``__pycache__`` directory should be excluded from the cache by the ``should_skip_path`` function.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

AI ASSESSMENT

Scenario: tests/test_fs.py::TestShouldSkipPath::test_skips_venv verifies that the function `should_skip_path` correctly identifies venv directories.

Why Needed: This test prevents a potential issue where the function `should_skip_path` incorrectly identifies venv directories as Python site packages, potentially leading to incorrect skipping of these directories.

Key Assertions:

- `assert should_skip_path('venv/lib/python/site.py') is True`
- `assert should_skip_path('.venv/lib/python/site.py') is True`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

AI ASSESSMENT

Scenario: Test the `_GeminiRateLimiter`'s pruning behavior when a request is added in the past.

Why Needed: This test prevents a potential issue where requests made before a certain time threshold are not properly cleared from the rate limiter's cache.

Key Assertions:

- The length of `_request_times` should be equal to 0 after pruning.
- The length of `_token_usage` should also be equal to 0 after pruning.
- The request times list should contain only one element (the time when the request was added).
- `_prune()` should not modify the request times list or token usage lists.
- The prune method should clear all requests and token usages from the rate limiter's cache.
- The test should pass without any assertion errors after pruning.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	11 lines (ranges: 39-42, 81-85, 87-88)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents requests from exceeding the specified limit.

Why Needed: This test prevents a potential bug where requests are allowed to exceed the specified rate limit, potentially leading to unexpected behavior or performance issues.

Key Assertions:

- The `wait` variable should be greater than 0.
- The `wait` variable should not exceed 60.0 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	26 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95, 97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents a regression when the token limit is exceeded.

Why Needed: This test verifies that the rate limiter correctly handles cases where the token limit is reached, preventing potential performance regressions.

Key Assertions:

- The next available time point should be greater than 0.
- The total number of tokens used since the last update should still be 2.
- The `_token_usage` list should contain only two elements.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	33 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 81-82, 84, 87-88, 92-94, 100-101, 103, 105, 107-108, 110-114)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the `wait_for_slot` method of `_GeminiRateLimiter` sleeps for a specified amount of time when a request is made.

Why Needed: This test prevents potential issues where requests are made too quickly and the rate limiter does not have enough time to process them.

Key Assertions:

- The `wait_for_slot` method should call `time.sleep` with the specified amount of time.
- The `wait_for_slot` method should assert that `mock_sleep` was called.
- The `wait_for_slot` method should not be able to make any requests while sleeping.
- The rate limiter's `record_request` method should have been called before making the request.
- The rate limiter's `record_request` method should have been called with a valid limit value.
- The rate limiter's `wait_for_slot` method should not be able to make any requests while waiting for the slot.
- The `time.sleep` function call should be within a reasonable delay (e.g. < 1 second).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	31 lines (ranges: 39-42, 45-46, 48, 52-54, 58-59, 61-63, 73, 76-78, 81-82, 84, 87-88, 92-93, 95, 97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the rate limiter records zero tokens when no tokens are available.

Why Needed: This test prevents a potential regression where the rate limiter does not record tokens for an extended period without reaching the limit.

Key Assertions:

- The `_token_usage` list of the `GeminiRateLimiter` instance is empty after calling `record_tokens(0)`.
- The `len(_token_usage)` property of the `GeminiRateLimiter` instance is equal to 0.
- The rate limiter's internal state is updated correctly when no tokens are available for recording.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	6 lines (ranges: 39-42, 66-67)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the test raises a RateLimitExceeded exception when exceeding daily limit.

Why Needed: This test prevents a regression where the rate limiter does not raise an error when exceeding the daily limit.

Key Assertions:

- The function ``wait_for_slot`` should raise ``_GeminiRateLimitExceeded`` with a message indicating that requests per day have exceeded the limit.
- The function ``record_request`` should be called before calling ``wait_for_slot`` to set up the rate limiter.
- The error message should include the string `'requests_per_day'` which is expected to be present in the match.
- The function ``wait_for_slot`` should not return immediately after raising the exception, but instead wait for the slot to become available.
- The function ``record_request`` should be called before calling ``wait_for_slot`` to set up the rate limiter.
- The error message should include the string `'requests_per_day'` which is expected to be present in the match.
- The function ``wait_for_slot`` should raise ``_GeminiRateLimitExceeded`` with a message indicating that requests per day have exceeded the limit.
- The function ``record_request`` should be called before calling ``wait_for_slot`` to set up the rate limiter.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	24 lines (ranges: 32-34, 39-42, 45-46, 48-50, 58-60, 73, 76-78, 81-82, 84, 87-88)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the rate limiter waits for TPM availability when tokens are used beyond the limit.

Why Needed: The test prevents a potential bug where the rate limiter does not wait for TPM availability even when tokens exceed the limit, leading to unexpected behavior.

Key Assertions:

- limiter._seconds_until_tpm_available(now, 5) > 0
- tokens_used + request_tokens > limit
- token_usage is not empty

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	24 lines (ranges: 39-42, 66, 68-70, 81-82, 84, 87-88, 100-101, 103, 105, 107-108, 110-114)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that RPM rate limit cooldown handling is properly implemented.

Why Needed: This test prevents a bug where the RPM rate limit cooldown is not set correctly on the first call to `_call_gemini`.

Key Assertions:

- The 'models/gemini-pro' model should be in the cooldowns dictionary with a value greater than 1000.0 seconds.
- The `provider._cooldowns['models/gemini-pro']` should have been set correctly after the first call to `_call_gemini`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	23 lines (ranges: 52-53, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	117 lines (ranges: 32-34, 39-42, 45-46, 48, 52-54, 66, 68-70, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-215, 217-223, 225-227, 233-234, 238-240, 242-243, 274-277, 280, 282-290, 292-295, 297-298, 300-301, 346, 348-350, 352-353, 381-382, 385-386)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the GeminiProvider's `_annotate_internal` method correctly handles rate limiting and retry logic when encountering a 429 status code.

Why Needed: This test prevents regression in the GeminiProvider class, ensuring it can handle cases where the API returns a 429 status code due to rate limiting.

Key Assertions:

- The 'annotation' object has the expected scenario ('Recovered Scenario') and number of calls to the 'mock_post' method (2).
- The 'annotation' object does not have an error attribute.
- The 'annotation' object's 'scenario' attribute is set to 'Recovered Scenario'.
- The 'annotation' object's 'error' attribute is None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	181 lines (ranges: 32-34, 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-215, 217-222, 225-227, 233-234, 238-240, 242-243, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330-333, 335-341, 343, 346, 348-350, 352-355, 360-363, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `_annotate_internal` method returns a valid `LlmAnnotation` object with the correct scenario and no error.

Why Needed: This test prevents regression where the `_annotate_internal` method fails to return an `LlmAnnotation` object due to incorrect response format from `_call_gemini`.

Key Assertions:

- The scenario of the annotation is set to 'Success Scenario'.
- No error is returned in the annotation. The error should be None.
- The annotation has a valid scenario.
- `_parse_response` returns a Mock object with the correct scenario and no error.
- `_call_gemini` returns text that matches the expected response format.
- The `_annotate_internal` method calls `_parse_response` correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	173 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-223, 225-227, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-343, 346, 348-350, 352-355, 360-363, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies that the availability check of a GeminiProvider instance returns False when no environment variables are set.

Why Needed: This test prevents a scenario where the availability check fails due to missing environment variables, potentially causing unexpected behavior or errors in downstream applications.

Key Assertions:

- The provider's `_check_availability()` method should return False for the given configuration.
- The provider's `_check_availability()` method should not throw an exception when no environment variables are set.
- The provider's `_check_availability()` method should correctly handle the case where environment variables are not present but a valid API token is provided.
- The provider's `_check_availability()` method should return True for the given configuration with a valid API token.
- The provider's `_check_availability()` method should throw an exception when no environment variables are set and a valid API token is not provided.
- The provider's `_check_availability()` method should correctly handle the case where environment variables are present but a valid API token is not provided.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	10 lines (ranges: 134, 136-139, 141-142, 266-267, 269)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify the rate limiter prevents a request from being blocked after reaching the limit.

Why Needed: This test prevents a potential issue where a user's requests are blocked due to exceeding the daily rate limit.

Key Assertions:

- The `next_available_in` method returns `None` when the limit has been reached.
- The limiter does not block any subsequent requests until the limit is reset.
- The limiter allows for at most one request per day, even if there are multiple requests in a short period.
- The limiter does not prevent users from making multiple requests within a short time frame.
- The limiter resets after each request, allowing for new requests to be made without blocking.
- The limiter does not block requests that have already passed the limit.
- The limiter allows for partial requests (e.g., 50% of the daily limit) to still be counted towards the total.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	18 lines (ranges: 39-42, 45-46, 48-50, 73, 76-78, 81-82, 84, 87-88)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the rate limiter does not block requests for a short period after the first two requests.

Why Needed: This test prevents a potential bug where the rate limiter blocks all subsequent requests for an extended period after the initial two requests, potentially causing unexpected behavior or errors.

Key Assertions:

- limiter.next_available_in(100) == 0.0
- limiter.record_request()
- assert limiter.next_available_in(100) == 0.0
- limiter.record_request()

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	27 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

- Scenario:** Test that different provider configurations result in different hash values.
- Why Needed:** This test prevents regression where the same configuration produces the same hash value, potentially due to a bug in the hashing algorithm or incorrect implementation of the Config class.
- Key Assertions:**
- The function `compute_config_hash(config)` should return a different hash for two different configurations (`config1` and `config2`).
 - The values returned by `compute_config_hash(config1)` and `compute_config_hash(config2)` should be distinct.
 - If the same configuration produces the same hash value, it may indicate an issue with the hashing algorithm or Config class implementation.
 - A different provider configuration should result in a different hash value for the same configuration.
 - The test should pass if the function correctly computes the hash of each configuration.
 - The test should fail if the function incorrectly computes the hash of one or both configurations.
 - If the test is run multiple times, it should produce different results each time due to the random nature of hashing.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 96-101, 103-104)

AI ASSESSMENT

Scenario: Verifies the length of the computed hash is exactly 16 characters.

Why Needed: This test prevents a potential issue where the hash might be too long, potentially causing issues with storage or transmission.

Key Assertions:

- The length of the computed hash should be exactly 16 characters.
- The hash value should not exceed 255 characters (the maximum allowed in Python).
- The hash value should not contain any non-ASCII characters.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 96-101, 103-104)

AI ASSESSMENT

Scenario: Verify that the computed SHA-256 hash of a file matches its content hash.

Why Needed: This test prevents regression where the file's contents are changed but the file hash remains consistent.

Key Assertions:

- The computed SHA-256 hash of the file should be equal to its content hash.
- The content hash calculated from the file's contents should match the expected value.
- The file hash should not change even if the file's contents are modified.
- The file hash should remain consistent across different runs of the test.
- The computed SHA-256 hash of a file with a known content hash should also be equal to that hash.
- The content hash calculated from a file with a known content hash should also match the expected value.
- The file hash should not change even if multiple files are written to the temporary directory.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	6 lines (ranges: 32, 44-48)

AI ASSESSMENT

Scenario: Verify the correctness of computing a SHA-256 hash for a file.

Why Needed: This test prevents potential issues where the computed hash does not match the expected output due to incorrect or missing file contents.

Key Assertions:

- The length of the computed hash should be 64 bytes.
- The computed hash should contain all characters from the input data (in this case, 'hello world').
- Any non-existent characters in the input data should not affect the computed hash.
- The computed hash should match the expected output provided by the `compute_file_sha256` function.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	5 lines (ranges: 44-48)

AI ASSESSMENT

Scenario: Verifying that different keys result in unique HMAC signatures.

Why Needed: This test prevents potential security vulnerabilities where the same key is used for multiple computations, potentially leading to predictable or reproducible signatures.

Key Assertions:

- The output of `compute_hmac(b'content', b'key1')` should be different from `compute_hmac(b'content', b'key2')`.
- The output of `compute_hmac(b'content', b'key3')` should not match either of the above outputs.
- The output of `compute_hmac(b'content', b'key1')` and `compute_hmac(b'content', b'key2')` should be different from each other.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 61)

AI ASSESSMENT

Scenario: Verify the length of the HMAC signature is correct.

Why Needed: This test prevents a potential issue where the HMAC signature might be too short or malformed, potentially causing errors in downstream processing.

Key Assertions:

- The length of the HMAC signature should be exactly 64 bytes.
- The HMAC signature should not be shorter than 32 bytes (the minimum required by most cryptographic standards).
- The HMAC signature should not exceed 128 bytes (the maximum allowed for most cryptographic applications).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 61)

AI ASSESSMENT

Scenario: The function `compute_sha256` is expected to produce the same hash for two identical input strings.

Why Needed: This test prevents a potential bug where different inputs could produce different hashes, potentially leading to inconsistent results.

Key Assertions:

- `h1 == h2` (assertion of equality)
- `h1.hex() == h2.hex()` (equality of hash values in hexadecimal format)
- `h1.digest() == h2.digest()` (equality of hash values as a bytes object)
- `compute_sha256(b'salt') != compute_sha256(b'other_salt')` (inequality of hashes for different inputs)
- `compute_sha256(b'test') == b'test'` (equality of hash value for the same input string)
- `compute_sha256(b'test2') != compute_sha256(b'test')` (inequality of hashes for different inputs)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 32)

AI ASSESSMENT

Scenario: Verify the length of the computed SHA-256 hash is 64 characters.

Why Needed: This test prevents a potential issue where the hash length may not be as expected, potentially leading to incorrect identification of the input data.

Key Assertions:

- The length of the hash should be exactly 64 hexadecimal characters.
- The hash string should contain all 256 possible hexadecimal digits (0-9, A-F, a-f).
- No padding bytes are present in the hash output.
- No leading zeros are present in the hash output.
- All characters in the hash output are hexadecimal digits (0-9, A-F, a-f).
- The hash is not empty.
- The hash does not contain any null bytes.
- The hash string contains only one character.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 32)

AI ASSESSMENT

Scenario: Verifies that the `get_dependency_snapshot()` function returns a snapshot including the 'pytest' package.

Why Needed: This test prevents a potential issue where the 'pytest' package is not included in the dependency snapshot, potentially causing issues with downstream dependencies.

Key Assertions:

- The 'pytest' package should be present in the snapshot.
- The 'pytest' package should be included in the snapshot's list of dependencies.
- The snapshot should contain a reference to the 'pytest' package.
- The snapshot should include all dependencies required by the 'pytest' package.
- The snapshot should not exclude any dependencies that are required by the 'pytest' package.
- The snapshot should be able to identify and report on the presence of the 'pytest' package.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 113-114, 116-121)

AI ASSESSMENT

Scenario: The `get_dependency_snapshot()` function should return a dictionary.

Why Needed: This test prevents a potential bug where the function returns an incorrect data type (e.g., a list instead of a dictionary).

Key Assertions:

- `snapshot` is an instance of `dict`
- `snapshot` has no attributes other than `__dict__`
- all keys in `snapshot` are strings
- no missing keys exist in `snapshot`
- `len(snapshot) == 0`
- `get_snapshot().keys() == set(['package1', 'package2'])`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 113-114, 116-121)

AI ASSESSMENT

Scenario: The test verifies that the `load_hmac_key` function correctly loads a key from a file.

Why Needed: This test prevents a bug where the `load_hmac_key` function fails to load a key from a file due to incorrect file path or permissions.

Key Assertions:

- The file path should be correct and accessible.
- The file should exist before attempting to load it.
- The file should not be empty or contain only whitespace characters.
- The `load_hmac_key` function should raise an error if the file is missing or corrupted.
- The `load_hmac_key` function should correctly read the key from the file and return it as expected.
- The `load_hmac_key` function should not throw any exceptions when loading a non-existent key.
- The `load_hmac_key` function should handle file paths with trailing slashes correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	5 lines (ranges: 73, 76-77, 80-81)

AI ASSESSMENT

Scenario: Test case: TestLoadHmacKey::test_missing_key_file verifies that the function returns None when a non-existent key file is provided.

Why Needed: This test prevents potential issues where the hmac_key_file parameter is not properly validated or checked for existence before attempting to load the HMAC key.

Key Assertions:

- The function should return None if the config object does not contain a 'hmac_key_file' attribute that points to an existing key file.
- The function should raise a ValueError with a descriptive message indicating that the key file is nonexistent when the config object does not contain a 'hmac_key_file' attribute.
- The function should check if the provided key file exists before attempting to load it using the Config class's hmac_key_file method.
- The function should handle the case where the key file is located in a different directory than expected (e.g., /home/user/.nonexistent.key).
- The function should not attempt to load an HMAC key from a non-existent or invalid key file, preventing potential security vulnerabilities.
- The function should provide informative error messages when the key file is missing or invalid, helping with debugging and troubleshooting.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	4 lines (ranges: 73, 76-78)

AI ASSESSMENT

Scenario: Verify that the function returns None when no key file is configured.

Why Needed: Prevents a potential bug where the function attempts to load an HMAC key without one being set.

Key Assertions:

- The `load_hmac_key` function should return `None` if no key file configuration is provided.
- The `Config` class should be able to create a default instance with no key file specified.
- The `load_hmac_key` function should raise an error when called without a valid key file configuration.
- The `key` variable should not hold any value in this case, as expected.
- The test should fail if the `load_hmac_key` function returns something other than `None`.
- The `assert` statement should raise an AssertionError with a meaningful message when the condition is not met.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	2 lines (ranges: 73-74)

AI ASSESSMENT

Scenario: Verifies that aggregation defaults to sensible settings.

Why Needed: Prevents a potential bug where aggregation is not configured correctly, leading to incorrect data retrieval or analysis.

Key Assertions:

- `config.aggregate_dir` is `None`
- `config.aggregate_policy == 'latest'`
- `config.aggregate_include_history` is `False`

COVERAGE

<code>src/pytest_llm_report/collector.py</code>	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
<code>src/pytest_llm_report/options.py</code>	3 lines (ranges: 107, 147, 233)
<code>src/pytest_llm_report/plugin.py</code>	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_integration_gate.py::TestConfigDefaults::test_capture_failed_output_default_false

1ms  3

AI ASSESSMENT

Scenario: Verify that the default capture failed output is set to False.

Why Needed: Prevents a regression where the default capture failed output is enabled by default.

Key Assertions:

- config.capture_failed_output should be False
- config.capture_failed_output is not True
- get_default_config().capture_failed_output is False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies that the context mode is set to 'minimal' by default.

Why Needed: This test prevents a potential bug where the context mode is not set to 'minimal' when no specific configuration is provided.

Key Assertions:

- config.llm_context_mode == 'minimal'
- get_default_config().llm_context_mode == 'minimal'
- assert isinstance(config, dict) and config.get('context_mode') == 'minimal'
- assert get_default_config().context_mode == 'minimal'
- assert get_default_config().context_mode != 'none' or get_default_config().context_mode != 'default'
- get_default_config().context_mode is not None
- get_default_config().context_mode != 'none'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that LLM is disabled by default in the configuration.

Why Needed: Prevent regression where LLM is enabled by default.

Key Assertions:

- The ``is_llm_enabled()`` method returns `False` for the default config.
- The ``is_llm_enabled()`` method should return `True` if LLM is not enabled by default.
- The configuration does not have a default value for ``llm_enabled`` set to `False`.
- The ``get_default_config()`` function returns an instance with a default value of `False` for ``llm_enabled``.
- The ``is_llm_enabled()`` method should raise an exception if the config is invalid or missing.
- The test case should fail when LLM is enabled by default in the configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	4 lines (ranges: 107, 147, 224, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the `TestConfigDefaults` class correctly sets `omit_tests_from_coverage` to `True` when `default` is set to `True`.

Why Needed: This test prevents a regression where the default behavior of omitting tests from coverage is not implemented correctly.

Key Assertions:

- The `TestConfigDefaults` class should set `omit_tests_from_coverage` to `True` when `default` is `True`.
- The value of `omit_tests_from_coverage` should be `True` for the given configuration.
- The test should not omit tests from coverage by default, even if `default` is `True`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_integration_gate.py::TestConfigDefaults::test_provider_de
fault_none

1ms



AI ASSESSMENT

Scenario: Tests the default provider setting when it is set to None.

Why Needed: Prevents a potential bug where the provider is not set to 'none' by default.

Key Assertions:

- The function `get_default_config()` returns an instance of `Config` with a provider attribute equal to 'none'.
- The `config.provider` property is accessed and compared to 'none'.
- An assertion error occurs if the `config.provider` is not 'none'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that secret files are excluded by default from the LLM context.

Why Needed: This test prevents a bug where secret files might be inadvertently included in the LLM context.

Key Assertions:

- The `llm_context_exclude_globs` configuration option is set to exclude 'secret' files.
- The `llm_context_exclude_globs` configuration option is set to exclude '.env' files.
- Any secret file names are present in the list of excluded globs.
- No secret file names are present in the list of excluded globs.
- The `llm_context_exclude_globs` configuration option is correctly set for LLM context.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test reports are deterministic (sorted by nodeid) and the output is correct.

Why Needed: This test prevents a regression where the order of reported tests changes due to changes in the `report_json` configuration.

Key Assertions:

- The `nodeids` list returned from `data['tests']` should be sorted in ascending order.
- Each `testid` in `data['tests']` should have a corresponding `nodeid` in the same order.
- All `nodeids` in `data['tests']` should be present in the output report.
- The number of unique `nodeids` in `data['tests']` should match the total number of tests.
- Each test result should have a corresponding `testid` and `nodeid` pair.
- All test results should have a valid outcome ('passed' or 'failed').
- The output report should contain all reported tests, sorted by nodeid.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	78 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	117 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Test that an empty test suite produces a valid report.

Why Needed: To prevent a regression where the test suite fails when there are no tests to run.

Key Assertions:

- The total count of tests in the report is zero.
- There is no summary data in the report.
- No error messages or warnings are present in the report.
- The report does not contain any failed tests.
- All test suites have been successfully written to the report.
- The report contains a valid JSON structure with the expected keys and values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	67 lines (ranges: 229-231, 233, 235, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	118 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: The test verifies that a full pipeline generates an HTML report.

Why Needed: This test prevents regression where the HTML report is not generated correctly due to a bug in the ReportWriter class.

Key Assertions:

- The file "report.html" exists at the specified path.
- The string '
- The keyword 'test_pass' is found in the content of the 'report.html' file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	113 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Test that the full pipeline generates a valid JSON report.

Why Needed: This test prevents regression in the integration gate, where it was previously possible to generate invalid or incomplete JSON reports.

Key Assertions:

- The report is generated with a valid schema version.
- The total count of tests is accurate (3 in this case).
- At least one test passed and at least one failed.
- At least one test was skipped.

COVERAGE

src/pytest_llm_report/_git_info.py	2 lines (ranges: 2-3)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	78 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)

src/pytest_llm_report/report_writer.py	133 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-322, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)
--	---

PASSED

tests/test_integration_gate.py::TestSchemaCompatibility::test_report_root_has_required_fields

1ms  3

AI ASSESSMENT

Scenario: Test that the ReportRoot has required fields when created with valid schema version, run meta and summary.

Why Needed: This test prevents a regression where a missing or invalid report root is created without specifying the required fields.

Key Assertions:

- The 'schema_version' field should be present in the data.
- The 'run_meta' field should be present in the data.
- The 'summary' field should be present in the data.
- The 'tests' field should be present in the data.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	54 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test 'RunMeta has aggregation fields' verifies that the test runs metadata includes aggregation fields.

Why Needed: This test prevents regression by ensuring RunMeta includes aggregation fields in its metadata.

Key Assertions:

- is_aggregated is present in data
- run_count is present in data

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test 'RunMeta has run status fields' verifies that the RunMeta object contains the necessary status fields.

Why Needed: This test prevents regression where the RunMeta object is missing or incorrectly configured status fields, potentially leading to incorrect interpretation of run metadata.

Key Assertions:

- The 'exit_code' field is present in the data.
- The 'interrupted' field is present in the data.
- The 'collect_only' field is present in the data.
- The 'collected_count' field is present in the data.
- The 'selected_count' field is present in the data.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_integration_gate.py::TestSchemaCompatibility::test_schema_version_defined

1ms



AI ASSESSMENT

Scenario: Verifies that the schema version is correctly defined and contains a dot (.) separating major and minor versions.

Why Needed: Prevents regression where the schema version is not properly defined or does not contain a valid dot separation.

Key Assertions:

- SCHEMA_VERSION should be set to a string value.
- SCHEMA_VERSION should contain at least one dot (.) separating major and minor versions.
- The presence of leading dots in SCHEMA_VERSION is not allowed.
- The presence of trailing dots in SCHEMA_VERSION is not allowed.
- The schema version should be a valid semver-like string.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The `TestSchemaCompatibility` function verifies that the `TestCaseResult` object has required fields.

Why Needed: This test prevents a potential bug where the `TestCaseResult` object is missing required fields, potentially leading to incorrect results or errors.

Key Assertions:

- `nodeid` should be present in the `data` dictionary
- `outcome` should be present in the `data` dictionary
- `duration` should be present in the `data` dictionary

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	17 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the `get_provider` function returns a `GeminiProvider` instance when the `provider` parameter is set to 'gemini'.

Why Needed: This test prevents a potential bug where the `get_provider` function incorrectly returns a different provider type when the `provider` parameter is not 'gemini'.

Key Assertions:

- The `__class__.__name__` attribute of the returned `GeminiProvider` instance should be equal to `'GeminiProvider'`.
- The `get_provider` function should return a `GeminiProvider` instance when the `provider` parameter is set to 'gemini'.
- The `get_provider` function should raise an error when the `provider` parameter is not set to 'gemini'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	10 lines (ranges: 52-53, 245, 247, 249, 252, 257, 262-263, 265)
src/pytest_llm_report/llm/gemini.py	7 lines (ranges: 134, 136-139, 141-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `get_provider` function returns a correct LiteLLMProvider instance when the provider is set to 'litellm'.

Why Needed: This test prevents regression where the LLM model is not correctly identified as 'LiteLLMProvider' even if it's being used with the correct provider.

Key Assertions:

- provider.__class__.__name__ should be equal to 'LiteLLMProvider'.
- get_provider(config).model == 'gpt-3.5-turbo'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	9 lines (ranges: 52-53, 245, 247, 249, 252, 257-258, 260)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: test_get_provider_with_none_provider returns NoopProvider.

Why Needed: The test prevents a potential bug where the LLM is not properly initialized with a None provider.

Key Assertions:

- provider should be None
- should return NoopProvider instance
- should have no dependencies

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 245, 247, 249-250)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that when using 'ollama' as a provider, OllamaProvider is returned.

Why Needed: This test prevents the regression of the previous one where 'ollama' was not correctly identified as an OllamaProvider.

Key Assertions:

- provider.__class__.__name__ should be equal to 'OllamaProvider'.
- The provider instance is an instance of OllamaProvider.
- The provider's class name matches the expected one (OllamaProvider).
- The provider is not None or empty.
- The provider has a valid model attribute.
- The provider does not have any invalid attributes.
- The provider does not raise any exceptions during initialization.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 245, 247, 249, 252-253, 255)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test case: Unknown provider raises ValueError when trying to retrieve a provider.

Why Needed: This test prevents the unknown provider from being used without proper configuration, ensuring that a ValueError is raised with a descriptive error message.

Key Assertions:

- The function `get_provider` should throw a ValueError when an unknown provider is specified.
- The error message should contain the string 'unknown'.
- The error message should be in lowercase to ensure case-insensitive matching.
- The error message should include the word 'unknown' explicitly.
- The test should fail with a ValueError exception when the `get_provider` function is called with an unknown provider.
- The test should not pass if the `get_provider` function returns an unknown provider without raising a ValueError.
- The test should verify that the error message is not empty and does not contain any other information.
- The test should be able to reproduce the issue on multiple runs of the test suite.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 245, 247, 249, 252, 257, 262, 267)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that NoopProvider implements LlmProvider contract.

Why Needed: Prevents regression in LlmProvider implementation.

Key Assertions:

- The provider should have the required methods: `annotate`, `is_available`, `get_model_name`, and `config`.
- The provider should not raise an exception when these methods are called.
- The provider should be able to access its configuration.
- The provider should be able to call its annotation method.
- The provider should be able to check if it's available.
- The provider should have a valid model name.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the annotate method returns an empty LlmAnnotation object when no annotation is provided.

Why Needed: This test prevents a regression where the NoopProvider does not return any annotation even if it has been configured with a config.

Key Assertions:

- annotation should be of type LlmAnnotation
- annotation scenario should be an empty string
- annotation why needed should be an empty string
- annotation key assertions should be an empty list

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the ``get_model_name`` method of the ``NoopProvider`` class returns an empty string when no model is specified.

Why Needed: This test prevents a regression where the ``get_model_name`` method would return a non-empty string for an empty input configuration.

Key Assertions:

- The ``get_model_name()`` method should return an empty string.
- The ``get_model_name()`` method should not throw any exceptions.
- The ``get_model_name()`` method should be able to handle the case where no model is specified in the configuration.
- The ``get_model_name()`` method should have a clear and consistent naming convention.
- The ``get_model_name()`` method should not modify the input configuration in any way.
- The ``get_model_name()`` method should return a string that accurately reflects the absence of a model name.
- The ``get_model_name()`` method should be able to handle different types of configurations (e.g. None, dict, etc.)
- The ``get_model_name()`` method should not throw any errors when given invalid input

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 66)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The NoopProvider should always be available in the tests.

Why Needed: This test prevents a potential regression where the NoopProvider might not be available due to an issue with the configuration or dependencies.

Key Assertions:

- provider.is_available() must return True
- config is not None
- provider is an instance of NoopProvider
- assert provider.is_available() is True

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 107, 110-111)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 58)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the annotation summary is printed when annotations run.

Why Needed: This test prevents a regression where the annotation summary is not printed, potentially causing confusion or errors.

Key Assertions:

- The function ``test_annotate_tests_emits_summary`` in ``tests/test_llm_annotator.py`` should print 'Annotated 1 test(s) via litellm' to the console.
- The function ``test_annotate_tests_emits_summary`` in ``tests/test_llm_annotator.py`` should capture and verify that this message is printed through the ``capsys.readouterr()`` method.
- The function ``test_annotate_tests_emits_summary`` in ``tests/test_llm_annotator.py`` should not fail or raise an exception if the annotation summary is not printed, ensuring test reliability.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	65 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

AI ASSESSMENT

- Scenario:** Test that the progress reporting callback is called for each test.
- Why Needed:** This test prevents a regression where the progress reporting callback might not be called for all tests, potentially causing issues with the overall test suite.
- Key Assertions:**
- The expected message should contain the correct information about the starting LLM annotation process.
 - The expected message should include the name of the test being annotated.
 - The expected message should display the progress report for a single test.
 - The progress report should indicate that all tests have been successfully annotated.
 - The progress report should not be empty after annotating all tests.
 - The progress report should contain the correct number of annotations (1 in this case).
 - The progress report should include the name of the test being annotated.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

AI ASSESSMENT

Scenario: Test that LLM annotations respect opt-out and limit settings.

Why Needed: This test prevents regression by ensuring LLM annotations do not skip opt-out tests or exceed the maximum allowed number of tests.

Key Assertions:

- Verify that only 'tests/test_a.py::test_a' is called when opt-out is enabled.
- Verify that no annotation is made for 'tests/test_b.py::test_b' with LLM_opt_out=True.
- Verify that the LLM annotator does not make any annotations for 'tests/test_c.py::test_c'.
- Verify that the number of tests annotated does not exceed 1 (LLM_max_tests=1).

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	65 lines (ranges: 45, 48-49, 56-57, 59, 61-62, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

AI ASSESSMENT

Scenario: Test that LLM annotations respect the requests-per-minute rate limit.

Why Needed: This test prevents a potential regression where the LLM annotator does not respect the requests-per-minute rate limit, leading to unexpected behavior or errors.

Key Assertions:

- The provider's calls should match the expected list of node IDs.
- The sleep function should be called twice with times equal to 2.0 seconds.
- The sleep function should not be called more than once per minute.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-173, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

AI ASSESSMENT

Scenario: Test that annotation skips unavailable providers with a clear message.

Why Needed: To prevent the annotation of tests from failing when an unavailable provider is detected.

Key Assertions:

- The test verifies that the annotation function skips the annotation process for unavailable providers.
- The test verifies that the annotation function returns a message indicating that the provider is not available.
- The test verifies that the captured output includes the message 'is not available' when the annotation process fails due to an unavailable provider.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Tests the use of cache in annotations.

Why Needed: This test prevents regression where the annotation process relies on a cached provider.

Key Assertions:

- The `provider` attribute of the `TestCaseResult` object is set to `tests/test_sample.py::test_case` after calling `annotate_tests`.
- The `llm_annotation` attribute of the `TestCaseResult` object is not `None` after calling `annotate_tests`.
- The scenario of the annotation process is still 'cached' even after calling `annotate_tests`.
- The provider is called when it should not be, indicating a regression in the cache behavior.

COVERAGE

src/pytest_llm_report/cache.py	30 lines (ranges: 39-41, 53, 55-56, 58, 60-62, 68-73, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-81, 87-92, 97-98, 100, 102, 104, 115-122, 127, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

PASSED

tests/test_llm_contract.py::TestAnnotationSchema::test_required_fields

1ms



AI ASSESSMENT

Scenario: The test verifies that the schema requires both 'scenario' and 'why_needed' fields.

Why Needed: This test prevents a potential bug where the schema is not enforced, allowing for invalid data to be accepted.

Key Assertions:

- assert 'scenario' in required
- assert 'why_needed' in required

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_llm_contract.py::TestAnnotationSchema::test_schema_from_dict

1ms



AI ASSESSMENT

Scenario: Test that AnnotationSchema.from_dict parses from a dictionary correctly.

Why Needed: Prevents data tampering by ensuring the correct parsing of schema fields from a dictionary.

Key Assertions:

- checks password
- checks username

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_llm_contract.py::TestAnnotationSchema::test_schema_handle
s_empty

1ms  3

AI ASSESSMENT

Scenario: The test verifies that the AnnotationSchema handles an empty input correctly.

Why Needed: This test prevents a potential bug where the AnnotationSchema may not be able to parse or validate an empty input.

Key Assertions:

- schema.scenario = ""
- schema.why_needed = ""

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the AnnotationSchema.from_dict method correctly handles a partial input scenario.

Why Needed: This test prevents regression where the AnnotationSchema.from_dict method does not handle partial inputs correctly.

Key Assertions:

- schema.scenario == 'Partial only'
- schema.why_needed == ''
- assert schema.scenario == 'Partial only' (to verify correct handling of partial input)
- assert schema.why_needed == '' (to verify absence of regression)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_llm_contract.py::TestAnnotationSchema::test_schema_has_required_fields

1ms



AI ASSESSMENT

Scenario: The test verifies that the schema has required fields.

Why Needed: This test prevents a bug where the schema is not properly defined with required fields, potentially leading to invalid or missing data.

Key Assertions:

- assert 'scenario' in ANNOTATION_JSON_SCHEMA['properties'],
- assert 'why_needed' in ANNOTATION_JSON_SCHEMA['properties'],
- assert 'key_assertions' in ANNOTATION_JSON_SCHEMA['properties']

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: TestAnnotationSchema::test_schema_to_dict verifies that the AnnotationSchema instance is correctly serialized to a dictionary.

Why Needed: This test prevents regression by ensuring that the AnnotationSchema instance can be converted into a dictionary without losing any critical information.

Key Assertions:

- assertion 1: The 'scenario' key in the dictionary matches the expected value.
- assertion 2: The 'why_needed' key in the dictionary matches the expected value.
- assertion 3: The 'key_assertions' list is present in the dictionary and contains all the expected values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 90-92, 94-96, 98)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Tests the factory method to return a NoopProvider when the provider is set to 'none'.

Why Needed: This test prevents a potential regression where a NoopProvider is returned for an invalid provider.

Key Assertions:

- The `provider` attribute of the `NoopProvider` instance should be `None`.
- The `get_provider` function should return an instance of `Config` with the correct provider set to `none`.
- The `isinstance(provider, NoopProvider)` assertion should pass for a `Config` object with the specified provider.
- The `provider` attribute of the returned `NoopProvider` instance should be `None`, not an instance of `NoopProvider`.
- The `get_provider` function should return an instance of `Config` with the correct provider set to `none` and a valid configuration.
- The `isinstance(get_provider(config), NoopProvider)` assertion should pass for a valid `Config` object with the specified provider.
- The `provider` attribute of the returned `NoopProvider` instance should be `None`, not an instance of `NoopProvider`.
- The `get_provider` function should return an instance of `Config` with the correct provider set to `none` and a valid configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 245, 247, 249-250)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The `NoopProvider` class is correctly instantiated as an instance of `LlmProvider`.

Why Needed: This test prevents a potential bug where the `NoopProvider` class might be incorrectly instantiated or not properly configured to implement the `LlmProvider` interface.

Key Assertions:

- The `provider` variable is assigned an instance of `Config`, which is expected to be used in conjunction with the `NoopProvider` class to create a valid LLM provider.
- The `provider` variable is assigned an instance of `LlmProvider`, which is the correct implementation of the interface.
- The `provider` variable has the correct type attribute set to `LLMProvider`.
- The `provider` variable does not have any additional attributes or methods that would prevent it from being used as a valid LLM provider.
- The `provider` variable does not have any invalid attributes or properties that could cause issues with its functionality.
- The `provider` variable is properly initialized and configured before use in the test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The NoopProvider returns an empty annotation when the test function does not contain any annotations.

Why Needed: This test prevents a regression where the NoopProvider's default behavior of returning an empty annotation is not applied in cases where the test function itself does not contain any annotations.

Key Assertions:

- The `annotate` method returns an empty annotation for a test that does not have any annotations.
- The `annotate` method should ideally return a meaningful annotation or error message when the test function contains no annotations.
- The test verifies that the `annotate` method correctly handles cases where the test function is empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the `annotate` method of the `ProviderContract` class returns an instance of `TestCaseResult` with the expected attributes.

Why Needed: This test prevents a potential regression where the `annotate` method does not return a valid `TestCaseResult` object, potentially causing issues downstream in the testing pipeline.

Key Assertions:

- The `scenario` attribute is present and has the correct value.
- The `why_needed` attribute is present and has the correct value.
- The `key_assertions` list contains all expected assertions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test Provider handles empty code with an empty contract.

Why Needed: This test prevents a potential bug where the provider does not handle cases with an empty contract.

Key Assertions:

- The `provider.annotate` method should return a non-None result for an empty contract.
- The `provider.annotate` method should set the `outcome` to 'passed' for an empty contract.
- An empty contract should be annotated correctly by the provider.
- A test with an empty code should pass without raising any errors or exceptions.
- No error message should be raised when annotating an empty contract.
- The provider's annotation process should not fail due to an empty contract.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the `provider` annotates a `TestCaseResult` with `None` when passed `None` as context.

Why Needed: This test prevents a potential bug where the provider does not handle cases with `None` context correctly, leading to incorrect results or errors.

Key Assertions:

- The `provider.annotate` method should return `None` when passed `None` as context.
- The `TestCaseResult` `nodeid` and `outcome` should be unchanged in this case.
- No exception should be raised when passing `None` as context to the `provider.annotate` method.
- The provider's annotation of the `TestCaseResult` should not affect its internal state or behavior.
- The test result should still pass even if the `provider` annotates an empty string instead of `None`.
- No error message should be printed when passing `None` as context to the `provider.annotate` method.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: All providers should have an annotate method.

Why Needed: This test prevents a potential bug where the LLM contract does not provide an annotate method for all providers.

Key Assertions:

- provider_name in ['none', 'ollama', 'litellm', 'gemini']
- hasattr(provider, 'annotate')
- callable(provider.annotate)
- provider_name is equal to 'none'
- provider_name is equal to 'ollama'
- provider_name is equal to 'litellm'
- provider_name is equal to 'gemini'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	15 lines (ranges: 52-53, 245, 247, 249-250, 252-253, 255, 257-258, 260, 262-263, 265)
src/pytest_llm_report/llm/gemini.py	7 lines (ranges: 134, 136-139, 141-142)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The `annotate` method of the `GeminiProvider` class is called with a context that exceeds its capacity.

Why Needed: This test prevents a potential issue where the `annotate` method might exceed its memory limit, leading to performance degradation or crashes.

Key Assertions:

- The `context` attribute of the `GeminiProvider` instance should not be larger than the maximum allowed size.
- The `annotation` function should not be called with an argument that is too large to fit in memory.
- The `annotate` method should raise a `MemoryError` exception when attempting to annotate a context that exceeds its capacity.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	9 lines (ranges: 52-53, 72, 75-76, 80, 165, 167, 175)
src/pytest_llm_report/llm/gemini.py	155 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-221, 233, 245-248, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 346, 348-350, 352-355, 360-363, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417-418, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The LiteLLM provider should report a missing dependency when the 'litellm' package is not installed.

Why Needed: This test prevents a bug where the provider incorrectly reports dependencies as installed when they are actually missing.

Key Assertions:

- The annotation error message should include the name of the missing dependency, which in this case is 'litellm'.
- The annotation error message should be informative and provide instructions on how to install the required package.
- The test case should fail when the 'litellm' package is not installed, but pass otherwise.
- The provider's behavior should change if the 'litellm' package is installed correctly, reporting an error instead of a success message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/gemini.py	12 lines (ranges: 134, 136-139, 141-142, 160-164)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `annotate` method throws an error when a missing API token is provided.

Why Needed: This test prevents a bug where the `GeminiProvider` class does not raise an error when an API token is missing, potentially leading to unexpected behavior or errors in downstream code.

Key Assertions:

- The `error` attribute of the annotation object should be equal to 'GEMINI_API_TOKEN is not set'.
- The `error` attribute of the annotation object should contain the string 'GEMINI_API_TOKEN is not set'.
- The `error` attribute of the annotation object should be a string.
- The `error` attribute of the annotation object should have the value 'GEMINI_API_TOKEN is not set'.
- The `error` attribute of the annotation object should contain the exact phrase 'GEMINI_API_TOKEN is not set'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/gemini.py	12 lines (ranges: 134, 136-139, 141-142, 160-161, 167-169)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that tokens recorded on the limiter are correctly annotated with usage metadata.

Why Needed: Prevents regressions where token usage is not properly reported.

Key Assertions:

- The 'totalTokenCount' in the response data should be equal to 123.
- The 'candidates' list should contain a single object with 'text' as its value.
- The 'usageMetadata' dictionary should contain a key named 'totalTokenCount'.
- The 'usageMetadata' dictionary should not be empty.
- The 'tokenUsage' list within the 'usageMetadata' dictionary should have exactly one element containing an integer value of 123.
- The 'candidates' list within the 'usageMetadata' dictionary should contain a single object with 'text' as its value and an integer value of 123.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	183 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-101, 103, 105, 107-109, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-223, 225-227, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-343, 346, 348-350, 352-355, 360-366, 368, 370-371, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)

src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The ``annotate`` method of the ``GeminiProvider`` class should retry when rate limiting occurs.

Why Needed: This test prevents a potential issue where the ``annotate`` method fails due to rate limiting and does not retry, leading to unexpected behavior.

Key Assertions:

- The ``annotate`` method should call ``self._retry_on_rate_limit`` before attempting to annotate.
- The ``annotate`` method should attempt to annotate again after a successful retry.
- The ``_retry_on_rate_limit`` method should be called with the correct arguments (e.g., ``max_retries``, ``rate_limit``, etc.)
- The ``annotate`` method should not raise an exception when rate limiting occurs, but instead retry the annotation operation
- The number of retries performed by the ``annotate`` method should increase accordingly as it attempts to annotate again after a successful retry
- The ``_retry_on_rate_limit`` method should be able to handle different types of rate limits (e.g., fixed, exponential)
- The ``annotate`` method should not fail unexpectedly when rate limiting occurs and retries are attempted

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	181 lines (ranges: 32-34, 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-222, 225-227, 233-234, 238-240, 242-243, 274-277, 280-283, 286, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330-333, 335-341, 343, 346, 348-350, 352-355, 360-366, 368-369, 374-377, 381-382, 385-387, 391-392, 396-399,

	401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The `annotate` method of the `GeminiProvider` class rotates models on the daily limit when called with a large number of annotations.

Why Needed: This test prevents a potential issue where the model rotation may not occur correctly if the number of annotations is too high, leading to inconsistent results.

Key Assertions:

- The `annotate` method should rotate all models on the daily limit when called with a large number of annotations.
- All models should be rotated after the specified number of annotations.
- No exceptions should be raised if the number of annotations is too high.
- The rotation of models should occur immediately after calling the `annotate` method.
- The rotation of models should not affect the performance of other methods in the class.
- The rotation of models should be consistent across different test runs with the same input data.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	177 lines (ranges: 39-42, 45-46, 48-50, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-208, 210, 213-214, 217-222, 225-227, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-341, 343, 346, 348-350, 352-355, 360-366, 368, 370, 372-377, 381-382, 385-387, 391-393, 396-399, 401-402, 405, 408-410, 412-414, 417, 419-420, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)

src/pytest_llm_report/options.py

2 lines (ranges: 107, 147)

src/pytest_llm_report/plugin.py

6 lines (ranges: 380-381,
384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the `annotate` method skips daily limits when used with a Gemini provider.

Why Needed: This test prevents a regression where the `annotate` method would incorrectly skip daily limits due to an incorrect implementation of the `GeminiProvider` class.

Key Assertions:

- The `annotate` method should not be called when the daily limit has been reached.
- The `annotate` method should return a specific error message indicating that daily limits have been exceeded.
- The `annotate` method should skip any subsequent calls to `annotate` for the same provider instance within a short period (e.g., 1 hour).
- The `annotate` method should not be called on providers with a 'daily_limit' key set to 0.
- The `annotate` method should raise an exception when called on providers with a 'daily_limit' key set to a negative value or zero.
- The `annotate` method should only skip calls for the same provider instance within a short period (e.g., 1 hour).
- The `annotate` method should not be called on providers that are being throttled or have their daily limit exceeded due to other factors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	184 lines (ranges: 39-42, 45-46, 48-50, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-208, 210-211, 213-214, 217-222, 225-227, 252-254, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-341, 343, 346, 348-350, 352-355, 360-366, 368, 370, 372-377, 381-382, 385-387, 391-393, 396-399, 401-402, 405, 408-410,

	412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the annotate method correctly annotates a successful response from LiteLLMProvider with mock response data.

Why Needed: Prevents regressions by ensuring that LiteLLMProvider returns a valid annotation even when it encounters a mock response.

Key Assertions:

- status ok
- redirect
- model gpt-4o
- content tests/test_auth.py::test_login
- def test_login()

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	177 lines (ranges: 39-42, 45-46, 48-49, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-101, 103, 105, 107-109, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-222, 225-227, 274-277, 280-283, 286, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-341, 343, 346, 348-350, 352-355, 360-366, 368-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)

src/pytest_llm_report/plugin.py

6 lines (ranges: 380-381,
384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the exhausted model recovers after 24 hours.

Why Needed: This test prevents a regression where the model does not recover from exhaustion within 24 hours.

Key Assertions:

- The model's performance should improve over time, and it should be able to handle a large number of queries without significant degradation.
- The model's memory usage should decrease after each query, indicating that it is recovering from exhaustion.
- The model's latency should decrease as the number of queries increases, further confirming recovery.
- The model's error rate should remain low or even decrease over time, suggesting a healthy recovery process.
- The model's ability to handle a large number of queries without significant degradation should be demonstrated through various metrics such as throughput and memory usage.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	190 lines (ranges: 39-42, 45-46, 48-50, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-188, 190-191, 193-194, 196, 200-208, 210-211, 213-214, 217-222, 225-227, 252-254, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-341, 343, 346, 348-350, 352-355, 360-366, 368, 370, 372-377, 381-382, 385-387, 391-393, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)

src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The `fetch_available_models` method of the `GeminiProvider` class raises an error when no models are available.

Why Needed: This test prevents a potential bug where the `fetch_available_models` method returns an error instead of raising one when there are no available models.

Key Assertions:

- `assertRaisesError`: The `fetch_available_models` method should raise an error when no models are available.
- `assertRaisesError`: The `fetch_available_models` method should not return any value (i.e., it should raise an exception).
- `assertRaisesValueError`: The `fetch_available_models` method should raise a `ValueError` with the message 'No models available'.
- `assertRaisesValueError`: The `fetch_available_models` method should raise a `ValueError` with the message 'No models available' when no models are available.
- `assertRaisesValueError`: The `fetch_available_models` method should not return any value (i.e., it should raise an exception).
- `assertRaisesValueError`: The `fetch_available_models` method should raise a `ValueError` with the message 'No models available'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	65 lines (ranges: 134, 136-139, 141-142, 280, 282-283, 286-290, 292-295, 297-298, 300-301, 346, 348-350, 352-355, 360-363, 374-377, 385, 387, 391-392, 396-402, 405, 408-410, 412-414, 417-418, 428, 430-432, 435-436)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The model list should refresh after an interval of time (e.g. seconds) when the LLM provider is updated.

Why Needed: This test prevents a potential regression where the model list does not update correctly after the LLM provider has been updated.

Key Assertions:

- The `refresh_model_list` method should be called with an interval of time (e.g. seconds) as its argument.
- The `refresh_model_list` method should return a new list of models without modifying the existing one.
- The number of models in the list should decrease by 1 after each update.
- The model list should contain only the updated models.
- The model list should not contain any deleted models.
- The LLM provider's `refresh_interval` attribute should be set to a valid value (e.g. 60 seconds).
- The `refresh_model_list` method should call the `update_models` method of the LLM provider with the correct arguments.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	169 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-222, 225-227, 274-277, 280-283, 286, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-341, 343, 346, 348-350, 352-355, 360-366, 368-369, 374-377, 381-382, 385-387, 391-393, 396-399, 401-402, 405, 408-410, 412-414, 417,

	419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_llm_providers.py::TestLiteLLMProvider::test_annotate_handles_completion_error

90.00s

5

AI ASSESSMENT

- Scenario:** The test verifies that the LiteLLMProvider annotates completion errors with a specific error message.
- Why Needed:** This test prevents regression by ensuring that the LiteLLM provider surfaces completion errors correctly.
- Key Assertions:**
- The annotation should contain the 'boom' error message.
 - The annotation should indicate that the function `test_case` was annotated with an error.
 - The annotation should include the specific error message 'boom'.
 - The annotation should not be empty or None.
 - The annotation should have a non-empty string value for the 'error' key.
 - The annotation should contain the correct type hint for the function `test_case`.
 - The annotation should indicate that the function `test_case` was annotated with an error.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175)
src/pytest_llm_report/llm/litellm_provider.py	22 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 78-79, 81-82, 84-85)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that LiteLLMProvider rejects invalid key_assertions payloads.

Why Needed: This test prevents regression where the provider incorrectly accepts non-list responses for key_assertions.

Key Assertions:

- response_data must be a list
- response_data must contain at least one 'key_assertion' item
- response_data must not contain any 'key_assertion' items with invalid types (e.g., string, dictionary)
- response_data must not contain any 'key_assertion' items with missing keys (e.g., no 'key' or 'assertion' key)
- response_data must not contain any 'key_assertion' items with duplicate keys
- response_data must not contain any 'key_assertion' items with invalid values (e.g., non-string, non-integer)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	22 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207-209)
src/pytest_llm_report/llm/litellm_provider.py	25 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 66-67, 69, 73, 76, 81-82, 84-85)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the LiteLLMProvider annotates a missing dependency correctly.

Why Needed: This test prevents a bug where the provider does not report an error for missing dependencies.

Key Assertions:

- The annotation includes the correct error message indicating that 'litellm' is missing and how to install it.
- The annotation includes the correct path to the installation command.
- The annotation includes the correct dependency name ('litellm').
- The annotation does not report an error for a non-existent dependency.
- The annotation reports the correct version of 'litellm' if available.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/litellm_provider.py	5 lines (ranges: 37-41)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the LiteLLM provider annotates a successful response with mock data.

Why Needed: Prevents regressions by verifying that the provider correctly handles successful responses.

Key Assertions:

- status ok
- redirect
- model gpt-4o
- tests/test_auth.py::test_login in messages

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/litellm_provider.py	20 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 66-67, 69-70)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the LiteLLM provider correctly detects the installed 'litellm' module.

Why Needed: This test prevents a potential bug where the provider does not detect the installed module.

Key Assertions:

- The function `provider.is_available()` should return `True` when the 'litellm' module is detected.
- The function `provider.is_available()` should raise an exception if the 'litellm' module is not detected.
- The function `provider.is_available()` should correctly handle cases where the 'litellm' module is installed but its path is incorrect.
- The function `provider.is_available()` should correctly handle cases where the 'litellm' module is installed but it does not have a valid import statement.
- The function `provider.is_available()` should raise an exception if the 'litellm' module has been removed from the system.
- The function `provider.is_available()` should correctly detect the 'litellm' module in Python 3.8 and later versions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 107, 110-111)
src/pytest_llm_report/llm/litellm_provider.py	3 lines (ranges: 94-95, 97)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The ``test_annotate_fallbacks_on_context_length_error`` test verifies that the LLM provider annotates fallbacks correctly when there is a context length error.

Why Needed: This test prevents regression in the LLM provider's behavior when encountering a context length error, ensuring consistent output for similar inputs.

Key Assertions:

- The ``annotate_fallback`` method returns an empty list when the input has no annotations.
- The ``annotate`` method returns an empty list when the input has no annotations and is not empty.
- The ``annotate_fallbacks_on_context_length_error`` test verifies that the LLM provider correctly annotates fallbacks in this scenario.
- The output of the ``annotate`` method for a context length error should be consistent across different inputs.
- The output of the ``annotate_fallbacks_on_context_length_error`` test should not contain any unexpected annotations.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	33 lines (ranges: 52-53, 72, 75-76, 78, 165, 167-173, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/ollama.py	15 lines (ranges: 40-41, 47, 50, 52, 54-55, 57-60, 62-63, 66-67)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test OllamaProvider::test_annotate_handles_call_error verifies that the annotate method returns an error message when a call to _call_ollama raises a RuntimeError.

Why Needed: This test prevents regression where the annotate method fails with an unexpected error message when a call to _call_ollama raises a RuntimeError.

Key Assertions:

- The annotation should return 'Failed after 10 retries. Last error: boom' as the error message.
- The annotation should not return any other error message or exception.
- The annotation should be able to handle multiple retries without changing the error message.
- The annotation should ignore the original call and only report the last error.
- The annotation should not raise an AssertionError when a call to _call_ollama raises a RuntimeError.
- The annotation should preserve the original system prompt.
- The annotation should be able to handle different types of errors (e.g., ConnectionError, TimeoutError).
- The annotation should not change the outcome of the test (i.e., it should still pass or fail based on the original call to _call_ollama)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175)
src/pytest_llm_report/llm/ollama.py	16 lines (ranges: 40-41, 47, 50, 52, 54-55, 57-59, 71-72, 74-75, 77-78)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The Ollama provider should report an error when annotating a test where the httpx dependency is missing.

Why Needed: This test prevents a potential bug where the provider incorrectly reports that httpx is installed when it's not, potentially leading to incorrect configuration or errors in downstream tests.

Key Assertions:

- The annotation should include an error message indicating that httpx is not installed.
- The error message should be specific to the missing dependency (httpx) and not generic.
- The provider should correctly report the missing dependency, rather than a misleading installation instruction.
- The test case should pass with this corrected annotation.
- The configuration of the OllamaProvider should be updated accordingly to reflect the correct state of dependencies.
- The error message should be logged by the provider and propagated to the user through the CaseResult object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 40-44)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the Ollama provider's full annotation flow with mocked HTTP.

Why Needed: Prevents authentication bugs by ensuring correct response from the API.

Key Assertions:

- Check status of the response
- Validate token in the response
- Verify that the response is not empty or None
- Assert that the response contains a specific JSON structure
- Check for any errors in the response
- Ensure that the response matches the expected model structure

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/ollama.py	29 lines (ranges: 40-41, 47, 50, 52, 54-55, 57-60, 62-63, 114, 116-123, 127-130, 132, 134-135)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test Ollama provider makes correct API call when calling OLLAMA successfully.

Why Needed: This test prevents regression where the OLLAMA provider fails to make a successful API call.

Key Assertions:

- The 'url' captured is set to the correct URL for the OLLAMA API call.
- The 'json' captured contains the expected response from the OLLAMA API.
- The 'timeout' captured matches the specified timeout value.
- The provider correctly calls the OLLAMA model with the provided parameters.
- The generated text is not empty and does not contain any errors.
- The stream parameter is set to False, indicating that no output will be produced.
- The API call has a successful status code (200).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	16 lines (ranges: 114, 116-123, 127-130, 132, 134-135)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the default model is used when not specified for Ollama provider.

Why Needed: This test prevents regression where the default model is not used, potentially causing unexpected behavior in downstream applications.

Key Assertions:

- The ``model`` attribute of the captured response is set to `'llama3.2'` (the default model).
- The ``json()`` method of the captured response returns a dictionary with a single key-value pair: ``{'response': 'ok'}``.
- The ``httpx.post()`` function from the ``FakeResponse`` class returns an instance of ``FakeResponse`` instead of a valid HTTP request object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	16 lines (ranges: 114, 116-123, 127-130, 132, 134-135)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the Ollama provider returns False when the server is unavailable.

Why Needed: This test prevents a regression where the provider incorrectly assumes the server is available even if it's not.

Key Assertions:

- The method `_check_availability()` of the `OllamaProvider` class should return `False` when the `get()` function raises a `ConnectionError`.
- The method `_check_availability()` of the `OllamaProvider` class should raise an `AssertionError` when the `get()` function raises a `ConnectionError`.
- The provider should not attempt to make requests to the server even if it's unavailable, as this would cause unexpected behavior.
- The provider should correctly handle the case where the server is down and return `False` without attempting to make any requests.
- The provider should raise an exception when the `get()` function raises a `ConnectionError`, rather than simply returning `False`.
- The provider should not silently fail or crash if it encounters a `ConnectionError`, but instead raise an exception that can be caught and handled by the test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	6 lines (ranges: 87-88, 90-91, 93-94)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the Ollama provider returns False for non-200 status codes.

Why Needed: This test prevents a potential bug where the provider incorrectly assumes all requests are successful (status code 200) when they may not be.

Key Assertions:

- The method `_check_availability()` of the provider should return False for a status code other than 200.
- The method `_check_availability()` of the provider should raise an exception if the status code is not 200.
- The provider's response object should have a 'status_code' attribute set to 500 (or any other non-200 status code).
- The provider's response object should not be None.
- The provider's response object should not have a 'status_code' attribute set to 200.
- The provider's response object should raise an exception if the status code is not 200 when calling `_check_availability()` method.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 87-88, 90-92)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test checks availability of Ollama provider via /api/tags endpoint.

Why Needed: Prevents regression in case the API is down or not responding correctly.

Key Assertions:

- The '/api/tags' URL should be present in the provided host.
- The response status code should be 200 (OK).
- The provider's check_availability method should return True.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 87-88, 90-92)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The Ollama provider should always return `is_local=True`.

Why Needed: This test prevents regressions where the provider might return `False` when it's actually local.

Key Assertions:

- `provider.is_local() == True`
- `provider.config.provider == 'ollama'`
- `config.config.provider == 'ollama'`
- `is_local()` is True in provider
- `is_local()` is True in config
- `provider.is_local()` should always be `True`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	1 lines (ranges: 102)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the `OllamaProvider` class throws an error when parsing a response with invalid JSON.

Why Needed: This test prevents a potential bug where the Ollama provider incorrectly reports valid responses as invalid JSON.

Key Assertions:

- The `annotation.error` attribute is set to 'Failed to parse LLM response as JSON'.
- The `provider._parse_response('not-json')` method returns an error message indicating that the response cannot be parsed as JSON.
- The test verifies that the error message contains the string 'Failed to parse LLM response as JSON'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 52-53, 186-187, 190-192)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-52, 55)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the OllamaProvider rejects invalid key_assertions payloads in its _parse_response method.

Why Needed: This test prevents a potential bug where the provider incorrectly handles invalid key_assertions payloads, potentially leading to unexpected behavior or errors.

Key Assertions:

- The response data must be a dictionary with a 'key_assertions' key
- The 'key_assertions' value must be a list of strings

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	16 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207-209)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The provided test verifies that the Ollama provider correctly parses a JSON response from a markdown code fence.

Why Needed: This test prevents regression in the LLM providers, ensuring they can extract relevant information from markdown code fences when parsing responses.

Key Assertions:

- `assert isinstance(response, dict)`
- `assert 'code_fence' in response`
- `assert response['code_fence'] is not None`
- `assert isinstance(response['code_fence'], str)`
- `assert len(response['code_fence']) > 0`
- `assert response['code_fence'].startswith('#')`
- `assert response['code_fence'].endswith('')`
- `assert 'annotation' in response`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	6 lines (ranges: 38, 42-44, 46-47)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The provided test verifies that the Ollama provider correctly extracts JSON from a plain markdown fence without a specified language.

Why Needed: This test prevents potential regression where the provider fails to extract JSON from such fences, potentially leading to incorrect output or errors.

Key Assertions:

- The response should contain a JSON object with the correct structure and keys.
- The response should have a 'data' key with the expected JSON data.
- The response should not be empty or null.
- The response should only contain JSON objects, not other types of content.
- The 'data' key should contain an array of objects with the correct structure.
- Each object in the 'data' array should have the correct keys and values.
- The provider should correctly handle nested objects and arrays.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	6 lines (ranges: 38, 42-44, 46-47)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the Ollama provider correctly parses valid JSON responses with expected assertions.

Why Needed: Prevents potential bugs in the LLM providers by ensuring they handle correct response data.

Key Assertions:

- assert a
- assert b

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Testing the `CoverageEntry` class to serialize correctly.

Why Needed: The test prevents a potential bug where the serialized data does not match the expected format.

Key Assertions:

- The 'file_path' key in the dictionary matches the expected value.
- The 'line_ranges' key in the dictionary matches the expected format.
- The 'line_count' key in the dictionary matches the expected value.
- The 'file_path' key is present and has the correct value.
- The 'line_ranges' key contains the correct ranges (1-3, 5, 10-15).
- The 'line_count' key contains the correct value (10).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	4 lines (ranges: 254-257)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that `CoverageEntry.to_dict()` correctly serializes a `CoverageEntry` object into a dictionary.

Why Needed: This test prevents the regression of coverage entry serialization issues in previous versions where the line ranges were not properly formatted.

Key Assertions:

- The 'file_path' key in the dictionary should be equal to 'src/foo.py'.
- The 'line_ranges' key in the dictionary should be equal to '1-3, 5, 10-15'.
- The 'line_count' key in the dictionary should be equal to 10.
- The line ranges are correctly formatted (i.e., separated by commas).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	3 lines (ranges: 207-209)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test coverage serialization for CoverageEntry.

Why Needed: This test prevents a bug where the CoverageEntry object's attributes are not properly serialized to JSON.

Key Assertions:

- The 'file_path' attribute is correctly set to 'src/foo.py'.
- The 'line_ranges' attribute is correctly set to '1-3, 5, 10-15'.
- The 'line_count' attribute is correctly set to 10.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	4 lines (ranges: 40-43)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: An empty annotation should be created with default values.

Why Needed: This test prevents a regression where an empty annotation would not have any attributes.

Key Assertions:

- `annotation.scenario == ""` (empty string)
- `annotation.why_needed == ""` (empty string) - This attribute is required for LlmAnnotation
- `annotation.key_assertions == []` (no assertions made in this class)
- `assert annotation.confidence is None` (default value for confidence)
- `assert annotation.error is None` (default value for error)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the `to_dict` method of `LlmAnnotation` returns a dictionary with required fields.

Why Needed: This test prevents regression where an annotation is missing some critical information.

Key Assertions:

- The 'scenario' key should be present in the dictionary.
- The 'why_needed' key should be present in the dictionary.
- The 'key_assertions' key should be present in the dictionary.
- The 'confidence' key should not be present in the dictionary when it is None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	8 lines (ranges: 104-107, 109, 111, 113, 115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: test_to_dict_with_all_fields verifies that the full annotation is correctly converted to a dictionary.

Why Needed: This test prevents potential issues where an incomplete or missing field in the annotation leads to incorrect data being returned from the `to_dict()` method.

Key Assertions:

- Asserts that the 'scenario' key in the resulting dictionary matches the expected value.
- Asserts that the 'confidence' key in the resulting dictionary is equal to the expected value (0.95).
- Asserts that the 'context_summary' key in the resulting dictionary has the correct mode and byte count values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	10 lines (ranges: 104-107, 109-111, 113-115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test default report functionality.

Why Needed: This test prevents a potential bug where the default report does not contain required schema version and empty lists for tests, collection errors, and warnings.

Key Assertions:

- The 'schema_version' key in the report dictionary should be equal to SCHEMA_VERSION.
- The 'tests' key in the report dictionary should be an empty list.
- The 'warnings' key in the report dictionary should not exist (i.e., its value is None or an empty string).
- The 'collection_errors' key in the report dictionary should not exist (i.e., its value is None or an empty string).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	54 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test Report with Collection Errors should include them.

Why Needed: This test prevents a regression where the report does not include collection errors.

Key Assertions:

- The 'collection_errors' key in the report dictionary should exist and have exactly one item.
- The 'nodeid' value of the first item in the 'collection_errors' list should match 'test_bad.py'.
- The 'message' value of the first item in the 'collection_errors' list should be 'SyntaxError'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	58 lines (ranges: 207-209, 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508-510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test verifies that the ReportRoot class correctly handles warnings in a report.

Why Needed: This test prevents a regression where reports with warnings are not properly handled.

Key Assertions:

- The 'warnings' key in the report dictionary should contain exactly one warning.
- The value of the 'code' key in the first warning should be 'W001'.
- The 'message' key in the first warning should match the provided message 'No coverage'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	60 lines (ranges: 229-231, 233, 235, 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Tests should be sorted by nodeid in output.

Why Needed: This test prevents a regression where the order of tests is not guaranteed to be consistent due to changes in the report generation logic.

Key Assertions:

- The list of nodeids in the reports should match the expected order.
- The nodeids should contain the correct values from the 'tests' dictionary.
- The nodeid 'a_test.py::test_a' should be present in the list.
- The nodeid 'm_test.py::test_m' should be present in the list.
- The nodeid 'z_test.py::test_z' should be present in the list and its value should match the expected order.
- All nodeids should contain unique values from the 'tests' dictionary.
- No duplicate nodeids should be present in the output.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	71 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test `test_to_dict_with_detail` verifies that the `ReportWarning` class's `to_dict()` method returns a dictionary with the 'detail' key.

Why Needed: This test prevents a potential warning about missing coverage details in reports.

Key Assertions:

- The 'detail' key should contain the path `'/path/to/file'`.
- The value of 'detail' is correct and matches the expected path.
- The 'ReportWarning' class's `to_dict()` method returns a dictionary with the required keys.
- The 'detail' key is present in the returned dictionary.
- The value of the 'detail' key is not empty or an empty string.
- The path `'/path/to/file'` is correctly formatted and matches the expected value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	6 lines (ranges: 229-231, 233-235)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test to dictionary without detail should exclude it.

Why Needed: Prevents a warning from being reported when the 'detail' key is missing from the report.

Key Assertions:

- The 'code' key should be present in the dictionary.
- The 'message' key should be present in the dictionary.
- The 'detail' key should not be present in the dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	5 lines (ranges: 229-231, 233, 235)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that RunMeta has aggregation fields.

Why Needed: Prevents regression where RunMeta is not aggregated and does not have necessary fields.

Key Assertions:

- The 'run_id' key in the meta dictionary should match the run ID provided.
- The 'run_group_id' key in the meta dictionary should match the run group ID provided.
- The 'is_aggregated' key in the meta dictionary should be True.
- The 'aggregation_policy' key in the meta dictionary should be set to 'merge'.
- The 'run_count' key in the meta dictionary should have a value of 3.
- The length of the 'source_reports' list in the meta dictionary should be equal to 2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	39 lines (ranges: 277-279, 281-283, 364-380, 382, 385, 387, 390, 393, 395, 397, 399-405, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that LLM fields are excluded when annotations are disabled.

Why Needed: Prevents regression where LLM fields are included even when annotations are disabled.

Key Assertions:

- The 'llm_annotations_enabled' key is not present in the data.
- The 'llm_provider' key is not present in the data.
- The 'llm_model' key is not present in the data.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test LLM traceability fields are included when enabled.

Why Needed: To ensure the correct inclusion of LLM traceability fields in the output.

Key Assertions:

- data['llm_annotations_enabled'] is True
- data['llm_provider'] == 'ollama'
- data['llm_model'] == 'llama3.2:1b'
- data['llm_context_mode'] == 'complete'
- data['llm_annotations_count'] == 10
- data['llm_annotations_errors'] == 2

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	40 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407-419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Testing the `to_dict()` method of `RunMeta` to ensure it excludes source reports when aggregated.

Why Needed: This test prevents a regression where non-aggregated runs are incorrectly included in source reports.

Key Assertions:

- The 'source_reports' key is not present in the dictionary returned by `to_dict()`.
- The value of `is_aggregated` is set to `False` when `to_dict()` is called on a non-aggregated run.
- Non-aggregated runs are correctly excluded from source reports.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test RunMeta to dict with all optional fields.

Why Needed: Prevents regression in case of missing or invalid optional fields.

Key Assertions:

- Verify that the 'git_sha' field is set correctly and is not empty.
- Verify that the 'git_dirty' field is True.
- Verify that the 'repo_version' field is set correctly and matches the provided value.
- Verify that the 'repo_git_sha' field is set correctly and matches the provided value.
- Verify that the 'repo_git_dirty' field is False.
- Verify that the 'plugin_git_sha' field is not present in the data.
- Verify that the length of the 'source_reports' list is 1 as expected.
- Verify that all required fields are present and have correct values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	49 lines (ranges: 277-279, 281-283, 364-380, 382-405, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test RunMeta to ensure it includes required run status fields.

Why Needed: This test prevents a potential bug where the `RunMeta` object is missing certain critical fields, potentially leading to incorrect or incomplete results.

Key Assertions:

- The 'exit_code' field should be equal to 1.
- The 'interrupted' field should be set to True.
- The 'collect_only' field should be set to True.
- The 'collected_count' field should match the expected value of 10.
- The 'selected_count' field should match the expected value of 8.
- The 'deselected_count' field should match the expected value of 2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify the schema version is in semver format.

Why Needed: Prevents regression where a non-semantic version string is used.

Key Assertions:

- The schema version should be split into three parts (e.g., '1.2.3').
- Each part of the version should consist only of digits (0-9).
- All parts of the version should be non-empty and not equal to zero.
- The first part of the version should be greater than 0.
- The second part of the version should be less than or equal to 99.
- The third part of the version should be less than or equal to 99.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `ReportRoot` class includes the schema version in its report root.

Why Needed: Prevents regression where the schema version is not included in reports.

Key Assertions:

- The `schema_version` attribute of the `ReportRoot` instance should be set to `SCHEMA_VERSION`.
- The `to_dict()` method of the `ReportRoot` instance should return a dictionary with a `schema_version` key equal to `SCHEMA_VERSION`.
- The value of the `schema_version` field in the report root JSON should match `SCHEMA_VERSION`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	54 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Tests CoverageEntry serialization correctly.

Why Needed: CoverageEntry does not serialize correctly if line ranges are missing or invalid.

Key Assertions:

- The 'file_path' key is present and matches the expected value.
- The 'line_ranges' key is present and matches the expected format.
- The 'line_count' key is present and matches the expected value.
- All assertions pass for a CoverageEntry with valid file path, line ranges, and line count.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	8 lines (ranges: 71-78)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the `to_dict` method of `LlmAnnotation` returns a dictionary with required fields.

Why Needed: This test prevents regression where an annotation is missing some critical information.

Key Assertions:

- The 'scenario' key should be present in the dictionary.
- The 'why_needed' key should be present in the dictionary.
- The 'key_assertions' key should be present in the dictionary.
- The 'confidence' key should not be present in the dictionary when it is None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	5 lines (ranges: 277-279, 281, 283)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test SourceReport to_dict_with_run_id verifies that the 'run_id' key is present in the resulting dictionary.

Why Needed: This test prevents a potential bug where the 'run_id' is missing from the output of the SourceReport.

Key Assertions:

- The 'run_id' key should be present in the dictionary.
- The value of the 'run_id' key should match the provided run_id.
- If no run_id is provided, the 'run_id' key should not be present in the dictionary.
- The 'run_id' key should have the correct format (e.g., 'run-1').
- Any errors or exceptions raised during the to_dict method execution should be caught and reported.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	6 lines (ranges: 277-279, 281-283)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the `CoverageEntry` object can be serialized to a dictionary correctly.

Why Needed: This test prevents a potential bug where the serialization of `CoverageEntry` objects may not work as expected, potentially leading to incorrect or missing information in the output.

Key Assertions:

- The 'file_path' key in the dictionary should match the actual file path.
- The 'line_ranges' key in the dictionary should match the provided line ranges.
- The 'line_count' key in the dictionary should match the expected value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	11 lines (ranges: 449-457, 459, 461)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that a minimal result has the required fields.

Why Needed: This test prevents regression where a minimal result is missing required fields.

Key Assertions:

- The 'nodeid' field should be set to the expected value.
- The 'outcome' field should be set to 'passed'.
- The 'duration' field should be set to 0.0 (indicating no execution time).
- The 'phase' field should be set to 'call'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	17 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test 'Result with coverage' verifies that the `CoverageEntry` object has a single entry in the `coverage` list.

Why Needed: This test prevents regression where the coverage report is not correctly formatted or contains duplicate entries.

Key Assertions:

- The `CoverageEntry` object has exactly one entry in the `coverage` list.
- The first element of the `coverage` list is a string representing the file path.
- Each `CoverageEntry` object in the `coverage` list has a `file_path` attribute matching the expected value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	22 lines (ranges: 40-43, 161-165, 167, 169, 171, 173, 176-178, 180, 182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `TestCaseResult` object includes a flag for LLM opt-out.

Why Needed: Prevents regression in case where LLM opt-out is enabled and the test passes.

Key Assertions:

- The value of `llm_opt_out` in the `d` dictionary should be `True`.
- The `TestCaseResult` object should have a `llm_opt_out` key with a boolean value of `True`.
- The `llm_opt_out` value should not be overridden by any other test case.
- If LLM opt-out is enabled, the test result should still pass even if the code under test fails.
- The `llm_opt_out` flag should be included in the output of the test.
- The `llm_opt_out` flag should be preserved across different test runs with LLM opt-out enabled.
- If LLM opt-out is disabled, the test result should still pass even if the code under test fails.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	18 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180-182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test 'test_result_with_rerun' verifies that the rerun fields are included in the TestCaseResult.

Why Needed: This test prevents regression where a result with reruns is not properly updated.

Key Assertions:

- The value of `rerun_count` should be equal to 2.
- The value of `final_outcome` should be 'passed'.
- The `rerun_count` field should be included in the TestCaseResult dictionary.
- The `final_outcome` field should be included in the TestCaseResult dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	19 lines (ranges: 161-165, 167, 169, 171, 173-176, 178, 180, 182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test case 'test_result_without_rerun_excludes_fields' verifies that the `result` dictionary does not include 'rerun_count' and 'final_outcome' keys.

Why Needed: This test prevents regression by ensuring that the `result` dictionary excludes fields related to reruns.

Key Assertions:

- The 'rerun_count' key is not present in the `result` dictionary.
- The 'final_outcome' key is not present in the `result` dictionary.
- The 'rerun_count' and 'final_outcome' keys are not included in the `result` dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	17 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify default values for Config object are set correctly.

Why Needed: Prevents regression in default configuration settings.

Key Assertions:

- `cfg.provider` should be 'none'
- `cfg.llm_context_mode` should be 'minimal'
- `cfg.llm_max_tests` should be 0
- `cfg.llm_max_retries` should be 10
- `cfg.llm_context_bytes` should be 32000
- `cfg.llm_context_file_limit` should be 10
- `cfg.llm_requests_per_minute` should be 5
- `cfg.llm_timeout_seconds` should be 30
- `cfg.llm_cache_ttl_seconds` should be 86400
- `cfg.include_phase` should be 'run'
- `cfg.aggregate_policy` should be 'latest'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies that the default configuration is correctly set to 'none'.

Why Needed: Prevents a potential bug where the default configuration is not properly initialized.

Key Assertions:

- The `cfg` variable should be an instance of `Config`.
- The `cfg.provider` attribute should be set to `None`.
- The `cfg` object should have no other attributes besides `provider`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the `'is_llm_enabled'` check returns `False` for a provider without an LLM.

Why Needed: To prevent regression in case of a change to the default provider or its configuration.

Key Assertions:

- The function `'Config.is_llm_enabled()'` should return `'False'` when the provider is set to `'none'`.
- The function `'Config.is_llm_enabled()'` should return `'True'` when the provider is set to `'ollama'`.
- The function `'Config.is_llm_enabled()'` should not return a value for an empty configuration object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test validates configuration with an invalid aggregate policy.

Why Needed: Prevents a potential bug where the aggregation policy is not properly validated and causes unexpected behavior or errors.

Key Assertions:

- The `aggregate_policy` parameter in the configuration should be one of 'sum', 'mean', 'min', 'max', 'count', 'stdev', 'median' or 'none'.
- If an invalid aggregate policy is passed, it should raise a validation error.
- The test should verify that only one error message is returned for an invalid aggregation policy.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-197, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Tests the validation of an invalid context mode.

Why Needed: Prevents a potential bug where the test fails due to an invalid context mode being used.

Key Assertions:

- The configuration object is created with an invalid context mode.
- An error message indicating the invalid context mode is returned.
- The 'Invalid llm_context_mode 'mega_max' error message is present in the list of errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-189, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Testing the `validate()` method with an invalid include phase.

Why Needed: Prevents a potential bug where an invalid include phase is not properly validated, causing unexpected behavior or errors.

Key Assertions:

- The `validate()` method should return at least one error message for an invalid include phase.
- The error message should contain the invalid include phase 'lunch_break'.
- The test should fail if no error messages are returned from the `validate()` method.
- The test should pass if exactly one error message is returned from the `validate()` method.
- The error message should not be empty or null.
- The error message should contain a string that can be used as an include phase.
- The error message should not contain any invalid characters.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-205, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test validation with an invalid provider.

Why Needed: Prevents a potential bug where the test fails due to an incorrect error message or count.

Key Assertions:

- The function `validate()` returns exactly one error message.
- The first error message contains the string 'Invalid provider'.
- The error message is not empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test validation of numeric constraints for TestConfig.

Why Needed: Prevents regression where the llm_context_bytes is set to a value less than 1000, potentially causing issues with LLM context creation.

Key Assertions:

- cfg.validate() should return at least 5 error messages
- llm_context_bytes must be at least 1000
- llm_max_tests must be 0 (no limit) or positive
- llm_requests_per_minute must be at least 1
- llm_timeout_seconds must be at least 1
- llm_max_retries must be 0 or positive

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	22 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209-218, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Valid configuration is validated successfully without any errors.

Why Needed: Prevents potential issues where invalid configurations are passed to the application.

Key Assertions:

- The `validate()` method of the `Config` class should return an empty list if the input configuration is valid.
- No exceptions should be raised when a valid configuration is provided.
- All required fields in the configuration should be present and have the expected values.
- All optional fields in the configuration should be absent or have default values.
- The `validate()` method should not throw any errors for well-formed configurations.
- Any invalid configurations should raise an exception with a meaningful error message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	17 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the `load_aggregation_options` function to ensure it correctly loads aggregation options from a mock Pytest configuration.

Why Needed: This test prevents regression in the `load_aggregation_options` function, which is responsible for loading aggregation options from a Pytest configuration.

Key Assertions:

- The `aggregate_dir` attribute of the loaded configuration should be set to `'aggr_dir'`.
- The `aggregate_policy` attribute of the loaded configuration should be set to `'merge'`.
- The `aggregate_run_id` attribute of the loaded configuration should be set to `'run-123'`.
- The `aggregate_group_id` attribute of the loaded configuration should be set to `'group-abc'`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	28 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286-294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test 'test_load_config_invalid_int_ini' verifies that the test loads configuration with invalid integer values in INI correctly.

Why Needed: This test prevents a potential bug where the test crashes or behaves unexpectedly when encountering invalid integer values in INI files.

Key Assertions:

- The function `load_config` should be able to handle and return valid default values for configuration options even if they are set to an invalid integer value.
- The function `mock_pytest_config.getini.side_effect` should not crash or throw an exception when called with an invalid key.
- The test should assert that the fallback value is correct (in this case, 10) and does not cause any unexpected behavior.
- The function `load_config` should be able to handle cases where the invalid integer value is set in a specific section of the INI file (e.g. 'llm_report_max_retries')
- The test should pass even if the invalid key is not found in the INI file, and the default value is used instead.
- The function `mock_pytest_config.getini` should return valid values for other configuration options that are not set to an invalid integer value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	28 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263-267, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the ``llm_coverage_source`` option is set to `'cov_dir'` when loading coverage source.

Why Needed: This test prevents a potential bug where the ``llm_coverage_source`` option is not correctly set to `'cov_dir'`.

Key Assertions:

- `mock_pytest_config.option.llm_coverage_source == 'cov_dir'`
- `cfg.llm_coverage_source == 'cov_dir'`
- `assert cfg.llm_coverage_source is None or cfg.llm_coverage_source == 'cov_dir'`
- `cfg.llm_coverage_source != 'cov_dir' and not isinstance(cfg.llm_coverage_source, str)`
- `cfg.llm_coverage_source != 'cov_dir' and isinstance(cfg.llm_coverage_source, dict)`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	25 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the `load_defaults` test loads the default provider and report HTML settings.

Why Needed: This test prevents a regression where the default provider or report HTML setting is not loaded when no options are set.

Key Assertions:

- `cfg.provider == 'none'`
- `cfg.report_html` is `None`

COVERAGE

<code>src/pytest_llm_report/collector.py</code>	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
<code>src/pytest_llm_report/options.py</code>	24 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
<code>src/pytest_llm_report/plugin.py</code>	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that CLI options override ini options.

Why Needed: This test prevents a regression where the CLI overrides ini settings, potentially causing unexpected behavior or errors.

Key Assertions:

- ini_value is set to 'cli_report.html' for llm_report_html option
- llm_requests_per_minute value is set to 100

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	27 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259-261, 263, 270-272, 274, 276, 278, 280-282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Testing the `load_from_cli` function with a specified maximum retries.

Why Needed: This test prevents potential regressions where the `llm_max_retries` option is not set to a valid value.

Key Assertions:

- The `llm_max_retries` option should be set to an integer value (e.g., 1, 3, or 9).
- The `load_config` function should return the expected configuration with the specified maximum retries.
- The `assert` statement should raise a `ValueError` if the `llm_max_retries` option is not set to an integer value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	25 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282-283, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test loading values from ini options.

Why Needed: Prevents a potential bug where the 'llm_report_provider' value is not set correctly in case of an error during configuration loading.

Key Assertions:

- The 'provider' key should be set to 'ollama'.
- The 'model' key should be set to 'llama3'.
- The 'context_mode' key should be set to 'balanced'.
- The 'requests_per_minute' key should be set to 10.
- The 'max_retries' key should be set to 5.
- The 'html' key should be set to 'report.html'.
- The 'json' key should be set to 'report.json'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	32 lines (ranges: 107, 147, 248, 251-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_options_extended.py::TestConfigAnnotations::test_aggregation_settings

1ms  3

AI ASSESSMENT

Scenario: Tests the aggregation settings configuration.

Why Needed: Prevents a potential bug where the aggregate policy is set to 'merge' without specifying an aggregate group ID, causing unexpected behavior in the aggregated reports.

Key Assertions:

- config.aggregate_dir should be equal to '/reports'.
- config.aggregate_policy should be equal to 'merge'.
- config.aggregate_include_history should be True.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_options_extended.py::TestConfigAnnotations::test_all_output_paths

1ms  3

AI ASSESSMENT

Scenario: Test Config with all output paths.

Why Needed: Prevents regression where the test is run without specifying all output paths.

Key Assertions:

- config.report_html == 'report.html'
- config.report_json == 'report.json'
- config.report_pdf == 'report.pdf'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Tests the `capture_failed_output` attribute of the `Config` class.

Why Needed: Prevents a potential bug where the test fails due to an incorrect configuration.

Key Assertions:

- The `capture_failed_output` attribute is set to True.
- The `capture_output_max_chars` attribute is set to 8000.
- The `capture_failed_output` attribute should be `True` if `capture_output_max_chars` is 8000 or more.
- If `capture_output_max_chars` is less than 8000, the test will fail due to an incorrect configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_options_extended.py::TestConfigAnnotations::test_compliance_settings

1ms  3

AI ASSESSMENT

Scenario: Verify that the `Config` object is created with the correct metadata file and HMAC key file.

Why Needed: This test prevents a bug where the configuration is not set correctly, potentially leading to incorrect compliance settings.

Key Assertions:

- The `metadata_file` attribute of the `Config` object is set to 'metadata.json'.
- The `hmac_key_file` attribute of the `Config` object is set to 'key.txt'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_options_extended.py::TestConfigAnnotations::test_coverage_settings

1ms  3

AI ASSESSMENT

Scenario: Test configures default coverage settings.

Why Needed: Prevents a regression where the test coverage is not enabled by default.

Key Assertions:

- config.omit_tests_from_coverage should be False
- config.include_phase should be 'all'
- config.omit_tests_from_coverage is set to False
- config.include_phase matches the expected value

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the ``llm_context_exclude_globs`` parameter is correctly set to exclude ``*.pyc`` and ``*.log`` files.

Why Needed: This test prevents a potential bug where the custom exclusion globs are not properly applied.

Key Assertions:

- The ``*.pyc`` glob matches the expected file extension.
- The ``*.log`` glob matches the expected file extension.
- The ``llm_context_exclude_globs`` parameter is set to include ``*.pyc`` and ``*.log`` files in the configuration.
- The custom exclusion globs are applied correctly without any additional files being included.
- No other files match the excluded globs.
- The ``*.pyc`` glob matches a file that does not exist (``non_existent_file.pyc``)
- The ``*.log`` glob matches a file that exists (``file.log``)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the ``llm_context_include_globs`` attribute includes only ``*.py`` files.

Why Needed: Prevents a potential bug where include globs are not correctly applied to LLM context.

Key Assertions:

- The ``*.py`` glob matches only files with a ``*.py`` extension.
- The ``*.pyi`` glob matches only files with a ``*.pyi`` extension.
- The ``llm_context_include_globs`` attribute is correctly updated with the specified globs.
- No other files are included in the LLM context.
- The include globs do not interfere with each other.
- The include globs are applied consistently across all test cases.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the `include_pytest_invocation` configuration option is set to `False`.

Why Needed: This test prevents a potential bug where the `include_pytest_invocation` option is not properly configured, potentially leading to unexpected behavior or errors in testing.

Key Assertions:

- The `include_pytest_invocation` attribute of the `Config` object is set to `False`.
- The `include_pytest_invocation` configuration option is not being used by the test.
- The `include_pytest_invocation` option has a default value of `True` in the `Config` class.
- The `include_pytest_invocation` attribute is being accessed and modified correctly within the test.
- The `config` object passed to the `Config` constructor has the correct attributes set.
- The `include_pytest_invocation` attribute is not being overridden by any other configuration options.
- The `include_pytest_invocation` attribute is being used in a valid way within the test.
- The `include_pytest_invocation` attribute is being updated correctly within the test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 107)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the LLM execution settings configuration.

Why Needed: Prevents regression in LLM execution settings configuration.

Key Assertions:

- The value of llm_max_tests is correctly set to 50.
- The value of llm_max_concurrency is correctly set to 8.
- The value of llm_requests_per_minute is correctly set to 12.
- The expected values are matched with the actual values in the config object.
- No other assertions are necessary for this test as it only verifies the LLM execution settings configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the configuration of LLM parameter settings.

Why Needed: This test prevents a potential bug where the LLM parameter values are not correctly configured, potentially leading to incorrect output or errors.

Key Assertions:

- The `llm_include_param_values` attribute is set to True.
- The `llm_param_value_max_chars` attribute is set to 200.
- The value of `llm_param_value_max_chars` is equal to 200.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the configuration of LLM settings for OLLAMA.

Why Needed: Prevents a potential bug where the model and context bytes are not set correctly.

Key Assertions:

- The `provider` attribute is set to 'ollama'.
- The `model` attribute is set to 'llama3.2'.
- The value of `llm_context_bytes` is equal to 64000.
- The value of `llm_context_file_limit` is not provided in the test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the `repo_root` attribute is correctly set to `/project` when a repository path is provided.

Why Needed: This test prevents a potential bug where the `repo_root` attribute is not set correctly if no explicit value is provided for the repository path.

Key Assertions:

- The `config.repo_root` attribute should be equal to `Path('/project')` when the `repo_root` parameter is passed to the `Config` constructor.
- The `repo_root` attribute of the test configuration object should match the expected value `/project`.
- If no explicit repository path is provided, the `repo_root` attribute should still be set correctly to `/project`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the `test_valid_phase_values` method to ensure all valid include_phase values pass validation.

Why Needed: This test prevents a potential bug where invalid or missing include_phase values cause the configuration to fail validation.

Key Assertions:

- The `validate()` method of the `Config` class should return an empty list of errors for each included phase (run, setup, teardown, all).
- Any error messages related to 'include_phase' should not be present in the validation results.
- All invalid or missing include_phase values should be ignored and not reported as errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	17 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that the default exclude globs are correctly set to include `*.pyc`, `__pycache__/*`, and any files with names containing `*secret*` or `*password*`.

Why Needed: This test prevents a potential bug where the default exclude globs do not match the expected patterns, potentially leading to issues when running the model.

Key Assertions:

- The function `*.pyc` is included in the default exclude globs.
- The function `__pycache__/*` is included in the default exclude globs.
- Any file with a name containing `*secret*` or `*password*` is included in the default exclude globs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Tests the default redact patterns configuration.

Why Needed: Prevents a potential security vulnerability by ensuring all sensitive information is redacted from default configurations.

Key Assertions:

- The function `Config().invocation_redact_patterns` returns a list of patterns that include `'--password'`, `'--token'`, and `'--api[_]?key'` to prevent sensitive data exposure.
- Any pattern found in the list should contain one of these keywords to ensure proper redaction.
- The presence of any other keyword should be checked for to prevent potential security issues.
- All patterns should match the expected format to guarantee correct redaction.
- No pattern should be missing any required keyword to maintain security standards.
- Any pattern with an invalid or incomplete format should raise an error to alert developers to fix it.
- The list of patterns should not contain any sensitive information that could be used for malicious purposes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Tests default configuration values.

Why Needed: To ensure correct default values are set for the test environment.

Key Assertions:

- config.provider should be set to 'none'.
- config.llm_context_mode should be set to 'minimal'.
- config.llm_context_bytes should be set to 32000 bytes.
- config.omit_tests_from_coverage should be True.
- config.include_phase should be set to 'run'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED

tests/test_options_maximal.py::TestConfigHelpersMaximal::test_is_llm_enabled

1ms  3

AI ASSESSMENT

Scenario: Verifies that the ``is_llm_enabled`` method returns the correct enabled status for different providers.

Why Needed: Prevents a regression where the method does not return `False` when LLM is disabled.

Key Assertions:

- The ``is_llm_enabled`` method should return `False` for provider `'none'`.
- The ``is_llm_enabled`` method should return `True` for providers `'ollama'`, `'litellm'`, and `'gemini'`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the validation of an invalid aggregate policy.

Why Needed: To prevent a potential bug where an invalid aggregate policy is used, which could lead to unexpected behavior or errors during configuration.

Key Assertions:

- The validate method returns at least one error for the given aggregate policy.
- The error message contains 'Invalid aggregate_policy' and specifies the specific aggregate policy as 'invalid'.
- An assertion is made that there is exactly one error in the list of errors returned by the validate method.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-197, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test validates the maximum options configuration with an invalid context mode.

Why Needed: Prevents a potential bug where the maximum options configuration is not validated correctly when an invalid context mode is provided.

Key Assertions:

- The 'llm_context_mode' field in the config should be set to one of the valid values (e.g. 'maximal', 'minimal')
- The error message for the invalid context mode should contain the exact phrase 'Invalid llm_context_mode 'invalid''
- At least one error should be returned when validating the configuration with an invalid context mode

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-189, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test the validation of an invalid include phase.

Why Needed: To prevent a potential bug where an invalid include phase is not properly validated and causes unexpected behavior.

Key Assertions:

- The function ``Config(include_phase='invalid')`` should return an error with a specific message.
- The error message should contain the string `'Invalid include_phase ' + 'invalid'` to identify the issue.
- The test should verify that there is only one error returned by the validation process.
- The error message should be present in the first assertion of the ``errors`` list.
- The error message should not be empty or null.
- The error message should contain the specified invalid include phase value.
- The function ``Config(include_phase='invalid')`` should raise an exception with a specific error message when called with an invalid include phase.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-205, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test validates an invalid provider.

Why Needed: Prevents a potential bug where the test fails due to an invalid provider being used.

Key Assertions:

- The configuration should return exactly one error for an invalid provider.
- The error message should contain 'Invalid provider 'invalid''.
- The error message should be present in the first error found by the validation process.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the Config class's validate method returns an error when numeric bounds are not met.

Why Needed: This test prevents a potential bug where invalid numeric values could be passed to the Config class, potentially causing unexpected behavior or errors in downstream systems.

Key Assertions:

- config.validate() should return at least 4 errors for invalid numeric values.
- The error messages should include 'llm_context_bytes', 'llm_max_tests', 'llm_requests_per_minute', and 'llm_timeout_seconds'.
- Each error message should contain the key 'llm_context_bytes', 'llm_max_tests', 'llm_requests_per_minute', or 'llm_timeout_seconds'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	21 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209-217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verifies that the ``validate`` method of a valid configuration returns an empty list.

Why Needed: Prevents a potential bug where an invalid configuration is passed to the ``validate`` method, causing it to return unexpected results or raise an exception.

Key Assertions:

- The ``validate`` method should be called on a valid configuration object and return an empty list.
- An empty list should be returned when the configuration is valid.
- Any exceptions raised by the ``validate`` method should be caught and reported as expected.
- The ``validate`` method should not throw any errors or warnings for valid configurations.
- The ``validate`` method should have a clear and consistent behavior for valid configurations.
- The ``validate`` method should only return an empty list when there are no invalid configuration values present.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	17 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the default configuration is loaded correctly.

Why Needed: This test prevents a potential bug where the default configuration is not properly loaded due to missing plugin options.

Key Assertions:

- The function `load_config(pytestconfig)` returns an instance of `Config`.
- The assertion `assert isinstance(cfg, Config)` checks if the returned object is indeed an instance of `Config`.
- The assertion `cfg = load_config(pytestconfig)` loads the configuration using the provided pytest config.
- The assertion `assert cfg is not None` ensures that a configuration is loaded even without any options registered.
- The assertion `cfg.get('plugin_options') == {}` checks if the plugin options are missing from the default configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	33 lines (ranges: 107, 147, 248, 251-259, 261, 263-265, 270, 272-276, 278, 280, 282, 286, 288, 290-292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Verify that markers in the plugin configuration exist.

Why Needed: Prevent a bug where markers are missing from the configuration.

Key Assertions:

- pytestconfig must be an instance of pytest.config.Config
- pytestconfig must have a 'markers' attribute
- The 'markers' attribute should contain all marker names
- A marker name must exist in the 'markers' list
- The 'markers' list must not be empty
- All marker names must exist in the 'markers' list

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the context marker does not cause errors in the LLM integration.

Why Needed: This test prevents regression and ensures that the LLM integration works correctly without causing any errors.

Key Assertions:

- The `test_llm_context_marker` function should pass without raising an error when run with a valid context marker.
- A message indicating that the context marker is not recognized or has been disabled should be displayed instead of raising an error.
- The LLM integration should work correctly and produce the expected output without any errors or warnings.
- Any exceptions raised during test execution should be caught and handled properly, rather than being propagated to the user.
- The `test_llm_context_marker` function should not raise any assertion errors when run with a valid context marker.
- A message indicating that the context marker is invalid or has been disabled should be displayed instead of raising an error.
- Any warnings raised during test execution should be caught and handled properly, rather than being propagated to the user.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the requirement marker does not cause any errors.

Why Needed: This test prevents a potential bug where the requirement marker could be misinterpreted as an error.

Key Assertions:

- The `requirement_marker` function should not raise any exceptions or throw any errors.
- The `requirement_marker` function should not modify the input data in any way that would cause unexpected behavior.
- The `requirement_marker` function should not have any side effects that could be misinterpreted as an error.
- The `requirement_marker` function should not take any arguments that are not relevant to its purpose.
- The `requirement_marker` function should return a value that indicates success or no error, rather than raising an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The test verifies that the report writer correctly generates a full report with both JSON and HTML output.

Why Needed: This test prevents regression where the report writer fails to generate a report for tests with multiple nodes or failed tests.

Key Assertions:

- The total number of tests passed should be 2 (test_a.py::test_pass) and 1 (test_b.py::test_fail).
- Both 'test_a.py' and 'test_b.py' should be included in the report HTML.
- The JSON output file should contain a summary with 'total' set to 2 and 'passed' set to 1.
- The HTML output file should include both 'test_a.py' and 'test_b.py'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	79 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)

src/pytest_llm_report/report_writer.py	131 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)
--	---

PASSED

tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_disabled

1ms  2

AI ASSESSMENT

Scenario: Test that collectreport skips when disabled and pytest_collectreport is mocked correctly.

Why Needed: To ensure that collectreport does not run with a false positive report when pytest_collectreport is disabled.

Key Assertions:

- mock_report.session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_collectreport.mock_report.session.config.stash.get.assert_called_with(_enabled_key, False)
- mock_report.session.config.stash.get.return_value == False
- pytest_collectreport.mock_report.session.config.stash.get.asserts_calledWith(_enabled_key, False)
- mock_report.session.config.stash.get.asserts_calledOnce
- pytest_collectreport.mock_report.session.config.stash.get.return_value != True

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	10 lines (ranges: 380-381, 384, 388-390, 401-402, 408-409)

PASSED

tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_enabled

2ms



AI ASSESSMENT

Scenario: Test that collectreport calls collector when enable is True.

Why Needed: The test prevents a potential regression where the plugin does not call the collector even if it's enabled.

Key Assertions:

- Mocking pytest_collectreport with a mock report object and stash_get function to verify collection report handling
- Asserting that collectreport calls collector once when _enabled_key is hit
- Verifying that collector handles collection report correctly

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	12 lines (ranges: 380-381, 384, 388-390, 401-402, 408, 412-414)

AI ASSESSMENT

Scenario: Verify that ``pytest_collectreport`` does not throw an exception when no session is available.

Why Needed: Prevent regression in plugin behavior when a session is not present.

Key Assertions:

- The function ``pytest_collectreport`` should not raise an exception when called with a mock report object without a session attribute.
- The mock report object should be able to be passed to ``pytest_collectreport`` without raising an exception.
- The test should fail if the ``session`` attribute is present in the mock report object but still raises an exception.
- The test should pass if the ``session`` attribute is not present in the mock report object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 380-381, 384, 388-390, 401, 405)

PASSED

tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_session_none

1ms  2

AI ASSESSMENT

Scenario: Verify that ``pytest_collectreport`` does not raise an exception when the session is ``None``.

Why Needed: Prevent regression in test cases where a ``None`` session is expected.

Key Assertions:

- The function ``pytest_collectreport`` should not be called with a ``None`` argument.
- No error should be raised when calling ``pytest_collectreport`` with a ``None`` session.
- The mock report object passed to ``pytest_collectreport`` should have a ``session`` attribute set to ``None``.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 380-381, 384, 388-390, 401, 405)

AI ASSESSMENT

Scenario: Test that LLM enabled warning is raised when using the ollama LLMS provider.

Why Needed: This test prevents a potential bug where the LLM report configuration is not properly validated and may lead to unexpected behavior or errors.

Key Assertions:

- The ``pytest_llm_report_provider`` option should be set to `'ollama'`.
- The ``llm_report_html``, ``llm_report_json``, ``llm_report_pdf``, ``llm_evidence_bundle``, ``llm_dependency_snapshot``, ``llm_requests_per_minute``, ``llm_aggregate_dir``, ``llm_aggregate_policy``, ``llm_aggregate_run_id``, and ``llm_aggregate_group_id`` options should be set to `None`.
- The ``llm_max_retries`` option should also be set to `None` for the LLM report configuration to work correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	44 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	29 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that validation errors raise UsageError when invalid configuration is provided.

Why Needed: This test prevents a potential bug where the plugin does not handle invalid configuration correctly and raises a UsageError instead of providing informative error messages.

Key Assertions:

- mock_config.getini.side_effect should be set to a lambda function that returns an exception with the correct message 'configuration errors'.
- mock_config.option.llm_report_html should be None.
- mock_config.option.llm_report_json should be None.
- mock_config.option.llm_report_pdf should be None.
- mock_config.option.llm_evidence_bundle should be None.
- mock_config.option.llm_dependency_snapshot should be None.
- mock_config.option.llm_requests_per_minute should be None.
- mock_config.option.llm_aggregate_dir should be None.
- mock_config.option.llm_aggregate_policy should be None.
- mock_config.option.llm_aggregate_run_id should be None.
- mock_config.option.llm_aggregate_group_id should be None.
- mock_config.option.llm_max_retries should be None.
- mock_config.rootpath should be set to a valid path.
- mock_config.stash should be an empty dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	43 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 248, 251-253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	25 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-199, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that configure skips on xdist workers when pytest_configure is called with a valid config.

Why Needed: This test prevents the 'pytest_configure' function from being called unnecessarily, which can lead to unexpected behavior or errors.

Key Assertions:

- The 'addinvalue_line' method of the mock_config object should not be called before the worker check is performed.
- The 'addinvalue_line' method of the mock_config object should only be called after the worker check has been performed.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	17 lines (ranges: 169-171, 173-175, 177-179, 183-184, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that fallback to load_config is triggered when Config.load is missing.

Why Needed: This test prevents a potential regression where the plugin fails to configure due to an empty Config object.

Key Assertions:

- Mocking `Config.getini` and `Config.option.llm_report_html` with `None` values ensures that `load_config` is called.
- The `validate` method of `Config` returns an empty list when `load_config` is called without a valid Config object.
- The `load_config` function is patched to return the mocked `MockConfig` instance.
- The `assert_called_once` method is used to verify that only one call to `load_config` occurs.
- The `mock_load.return_value` attribute is set to `MockConfig` to ensure that it is called with the mocked Config object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	29 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test loading all INI options in the plugin configuration.

Why Needed: This test prevents regression where the plugin fails to load INI options when CLI options are not set.

Key Assertions:

- The correct provider is 'ollama'.
- The correct model is 'llama3.2'.
- The correct context mode is 'complete'.
- The correct number of requests per minute is 10.
- The correct report HTML file name is 'ini.html'.
- The correct report JSON file name is 'ini.json'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	31 lines (ranges: 107, 147, 248, 251-263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test CLI options override INI options.

Why Needed: This test prevents a potential regression where the CLI options override INI options, potentially causing unexpected behavior or incorrect results.

Key Assertions:

- Verify that `llm_report_html` is set to `cli.html` in the configuration.
- Verify that `llm_report_json` is set to `cli.json` in the configuration.
- Verify that `report_pdf` is set to `cli.pdf` in the configuration.
- Verify that `report_evidence_bundle` is set to `bundle.zip` in the configuration.
- Verify that `report_dependency_snapshot` is set to `deps.json` in the configuration.
- Verify that `llm_requests_per_minute` is set to 20 in the configuration.
- Verify that `aggregate_dir` is set to `/agg` in the configuration.
- Verify that `aggregate_policy` is set to `merge` in the configuration.
- Verify that `aggregate_run_id` is set to `run-123` in the configuration.
- Verify that `aggregate_group_id` is set to `group-abc` in the configuration.
- Verify that the root path of the configuration is `/project` in the configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	38 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259-263, 270-283, 286-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that terminal summary skips when plugin is disabled.

Why Needed: This test prevents a regression where the terminal summary is not properly skipped when the plugin is disabled.

Key Assertions:

- The `pytest_terminal_summary` function should be called with an empty stash and no worker input.
- The `stash.get()` method of the mock configuration object should have been called once with `_enabled_key` as its argument and `False` as its value.
- The `stash.get()` method of the mock configuration object should not have been called again for subsequent assertions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	9 lines (ranges: 238, 242-243, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that terminal summary skips on xdist worker when a specific configuration is used.

Why Needed: This test prevents regression in the plugin's behavior when using an xdist worker with a specific configuration.

Key Assertions:

- The function `pytest_terminal_summary` should return `None` for the given mock config.
- The function `pytest_terminal_summary` should not perform any actions on the given mock config.
- The function `pytest_terminal_summary` should not call any functions or methods on the given mock config.
- The function `pytest_terminal_summary` should not modify the given mock config in any way.
- The function `pytest_terminal_summary` should not raise any exceptions when called with a mock config.
- The function `pytest_terminal_summary` should behave as expected when called with a mock config that is different from the original configuration.

COVERAGE

<code>src/pytest_llm_report/collector.py</code>	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
<code>src/pytest_llm_report/plugin.py</code>	8 lines (ranges: 238-239, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test config loading from pytest objects (CLI + INI) to ensure it correctly sets the report HTML.

Why Needed: This test prevents a potential bug where the report HTML is not set correctly if the `pytest_llm_report` options are missing or invalid.

Key Assertions:

- The `report_html` attribute of the loaded configuration object should be set to 'out.html'.
- If `pytest_llm_report.option.llm_report_json` is set, it should not affect the value of `report_html`.
- If `pytest_llm_report.option.llm_report_html` is set, it should override any previously set `report_html`.
- The `rootpath` attribute of the loaded configuration object should be set to '/root'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	36 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270-283, 286-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test makereport skips when disabled.

Why Needed: This test prevents a regression where the plugin's makereport functionality is not properly handled when the 'pytest_runtest_makereport' hook is disabled.

Key Assertions:

- The ``mock_item.config.stash.get`` call returns ``False`` instead of raising an error when it should be ``None``.
- The ``mock_call`` object does not raise a ``StopIteration`` exception when the generator completes normally.
- The ``gen.send(mock_outcome)`` call raises a ``StopIteration`` exception when the generator completes normally, but this is not handled correctly by the plugin.
- The plugin's makereport functionality is not properly cleaned up when the 'pytest_runtest_makereport' hook is disabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	7 lines (ranges: 380-381, 384-385, 388-390)

AI ASSESSMENT

Scenario: Test makereport calls collector when enabled.

Why Needed: Prevents a potential bug where the plugin does not call the collector when makereport is enabled.

Key Assertions:

- The `pytest_runtest_makereport` function should be called with the `mock_collector` instance as its second argument.
- The `mock_collector.handle_runtest_logreport` method should be called with the `mock_report` instance and `mock_item` instance as arguments.
- The `mock_collector` instance should have a `handle_runtest_logreport` method that takes two arguments: `mock_report` and `mock_item`.
- The `mock_collector` instance should be able to handle the `runtest_logreport` event with the correct arguments.
- The `mock_collector.handle_runtest_logreport` method should not raise an exception when called with a mock report object.
- The `pytest_runtest_makereport` function should yield control back to the test suite after calling the collector.
- The `mock_collector` instance should be able to handle multiple calls to `handle_runtest_logreport` without raising any exceptions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that collection_finish is skipped when disabled.

Why Needed: To prevent a regression where the plugin's hooks are not executed correctly when collection finish is disabled.

Key Assertions:

- The pytest_collection_finish function should be called with False as the stash.get argument.
- The pytest_collection_finish function should have been called once with _enabled_key and False as its argument.
- The mock_session.config.stash.get method should have returned False when it was called with _enabled_key and False as its argument.
- The pytest_collection_finish function should not have executed any hooks in this test case.
- The pytest_collection_finish function should not have been called multiple times in this test case.
- The pytest_collection_finish function should only be called once in this test case.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 380-381, 384, 388-390, 424-425)

PASSED

tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_enabled

2ms  2

AI ASSESSMENT

Scenario: Test that collection_finish is called when collection finish is enabled.

Why Needed: This test prevents a potential bug where the collector is not called when collection finish is enabled.

Key Assertions:

- The stash_get function should return True for _enabled_key and False for _collector_key.
- The mock_collector handle_collection_finish method should be called once with mock_session.items as argument.
- MockSession items should have exactly two elements.
- MockCollector should not call stash_get or handle_collection_finish on other keys.
- pytest_collection_finish should be called with mock_session.items as argument when collection finish is enabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	10 lines (ranges: 380-381, 384, 388-390, 424, 428-430)

PASSED

tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_disabled

1ms  2

AI ASSESSMENT

Scenario: Test that sessionstart skips when disabled and checks enabled status.

Why Needed: This test prevents a regression where pytest_sessionstart fails to check the plugin's enabled status when the session is started in disabled mode.

Key Assertions:

- mocked config.stash.get was called with _enabled_key and False
- mocked config.stash.get was not called with _enabled_key or False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 380-381, 384, 388-390, 441-442)

AI ASSESSMENT

Scenario: Verify that sessionstart initializes collector when enabled.

Why Needed: Prevents a potential bug where the collector is not initialized even though pytest_sessionstart is enabled.

Key Assertions:

- The _collector_key should be present in mock_stash.
- The _start_time_key should be present in mock_stash.
- The stash dictionary should contain _enabled_key set to True.
- The stash dictionary should not have _config_key set to None.
- The stash dictionary should support get() and [] operations without raising an error.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	11 lines (ranges: 380-381, 384, 388-390, 441, 445, 448, 450-451)

AI ASSESSMENT

Scenario: Test `pytest_adoption` adds expected arguments and verifies specific options.

Why Needed: `pytest_adoption` may not be correctly handling the 'llm-report' option or other related arguments.

Key Assertions:

- `parser.getgroup.assert_called_with('llm-report', 'LLM-enhanced test reports')`
- `calls.any('--llm-report' in args[0])`
- `calls.any('--llm-coverage-source' in args[0])`

COVERAGE

<code>src/pytest_llm_report/collector.py</code>	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
<code>src/pytest_llm_report/plugin.py</code>	99 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test `pytest_addoption` adds INI options (lines 13-34) to ensure it correctly handles ini file additions.

Why Needed: This test prevents regression where `pytest_addoption` does not add INI options to the plugin's configuration.

Key Assertions:

- The function `pytest_addoption(parser)` is called with a `MagicMock` instance as its argument.
- The `addini.call_args_list` attribute of the `parser` object is checked for the expected ini file additions.
- The INI options `'llm_report_html'`, `'llm_report_json'`, and `'llm_report_max_retries'` are found in the ini calls.
- The expected values are verified to be present in the ini calls.
- The `MagicMock` instance is used instead of a real parser object, ensuring the test can run independently.
- The test does not rely on any specific plugin configuration or setup.

COVERAGE

<code>src/pytest_llm_report/collector.py</code>	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
<code>src/pytest_llm_report/plugin.py</code>	99 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test coverage percentage calculation logic for terminal summary.

Why Needed: Prevents regression in coverage reporting when terminal summaries are generated.

Key Assertions:

- The `pytest_terminal_summary` function correctly calculates the coverage percentage.
- The `CoverageMapper` class is properly loaded and used to report coverage.
- The `ReportWriter` class is called with the correct arguments.
- Mocking the existence of a coverage file does not prevent the test from running.
- The `report` method of the Coverage object is called correctly.
- The `load` method of the CoverageMapper class is called correctly.
- The `report_html` parameter is set to 'out.html' as expected.
- The mock coverage report matches the expected value (85.5%) when run with a valid configuration.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	53 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-305, 307, 309-312, 324-325, 330-331, 358-368, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test terminal summary with LLM enabled runs annotations.

Why Needed: This test prevents regression in the case where LLM is enabled and the plugin provider is not properly configured.

Key Assertions:

- Verify that `pytest_terminal_summary_llm_enabled` is called with the correct configuration.
- Check if the config passed to `pytest_terminal_summary_llm_enabled` is correctly set.
- Assert that the annotation is called only once, even when multiple tests are run in parallel.
- Verify that the correct model name is used for LLM-based providers.
- Ensure that the provider is properly configured before running the test.
- Test that the `pytest_terminal_summary_llm_enabled` function handles different scenarios correctly.
- Verify that the mock stash is created and populated correctly with the provided configuration.
- Check if the coverage map is not affected by the patching of dependencies.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	59 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324-325, 330-333, 336, 338, 341-343, 350-355, 358-368, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test terminal summary creates collector if missing.

Why Needed: This test prevents a regression where the plugin does not create a collector even when it is supposed to be present in the configuration.

Key Assertions:

- `mock_terminalreporter.call_args_list[0][1].get(_enabled_key) == False`
- `mock_terminalreporter.call_args_list[0][1].get(_config_key).report_html == 'out.html'`
- `mock_stash._enabled_key == True`
- `mock_stash._config_key == cfg`
- `mock_writer_cls.return_value.__call__.return_value.report_html == 'out.html'`
- `mock_mapper.map_coverage.return_value == {}`
- `mock_mapper_cls.return_value.__call__.return_value.map_coverage.return_value == {}`

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	45 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test terminal summary with aggregation enabled.

Why Needed: Prevents a regression where the aggregation feature is not properly handled in the terminal summary.

Key Assertions:

- The ``aggregate_dir`` parameter should be set to ``/agg`` when using the ``terminal_summary`` function with aggregation.
- The ``aggregate_report`` method of the `Aggregator` instance should be called once with a report object.
- The ``write_json`` and ``write_html`` methods of the `ReportWriter` instance should be called once with the correct data.
- The ``aggregate_dir`` parameter should not be set to ``/agg`` when using the ``terminal_summary`` function without aggregation.
- The ``aggregate_report`` method of the `Aggregator` instance should not be called if no report is provided.
- The ``write_json`` and ``write_html`` methods of the `ReportWriter` instance should not be called if no data is provided.
- The ``aggregate_dir`` parameter should be set to ``/agg`` when using the ``terminal_summary`` function with aggregation, regardless of the number of runs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	21 lines (ranges: 238, 242, 246, 249-250, 252-253, 256-257, 259, 261-265, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test coverage calculation error when loading coverage map.

Why Needed: To prevent a potential bug where the coverage calculation fails due to an OSError during load of the coverage map.

Key Assertions:

- The `load` method of the `CoverageMapper` class should not raise an exception when the coverage map is loaded successfully.
- The `coverage.Coverage` object returned by `CoverageMapper.load()` should be a valid instance.
- The `report_writer.ReportWriter` object created with `ReportWriter` should not raise an exception when writing to it.
- The `pytest_terminal_summary` function should not raise a `UserWarning` when called with the mock configuration and coverage map.
- The `coverage.Coverage` object returned by `CoverageMapper.load()` has a valid `report_html` attribute.
- The `coverage.Coverage` object returned by `CoverageMapper.load()` has a valid `stash` attribute.
- The `pytest_terminal_summary` function does not raise an exception when called with the mock configuration and coverage map.
- The `pytest_terminal_summary` function writes to the report writer without raising an exception.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	52 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-305, 315-318, 324-325, 330-331, 358-368, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Tests the ContextAssembler to assemble a balanced context for test_a.py with a utility function.

Why Needed: This test prevents regression that may occur when the llm_context_mode is set to 'balanced' and the assembly of the context fails due to missing dependencies.

Key Assertions:

- The test verifies that the assembled context includes the required file utils.py.
- The test verifies that the assembled context includes the required function def util() from the required file utils.py.
- The test ensures that the required file utils.py is present in the assembled context.
- The test verifies that the required function def util() is included in the assembly of the context.
- The test checks for coverage of the required file utils.py and function def util() within the assembly of the context.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	51 lines (ranges: 33, 49, 52, 55, 58, 60-61, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 116, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-155, 158-159, 163, 191-192, 194)

AI ASSESSMENT

Scenario: The `ContextAssembler` should assemble the complete context for a test file with no imports.

Why Needed: This test prevents regression when a test file has no imports, as it ensures that the context is correctly assembled even without external dependencies.

Key Assertions:

- The source code of the test function `test_1` should be present in the assembled context.
- The `test_1` function should be found in the assembled context.
- The `test_a.py::test_1` nodeid should match the actual file path of the test function.
- The `ContextAssembler` should correctly assemble a test file with no imports.
- The `TestCaseResult` nodeid should contain the correct information about the test result.
- The context should not be empty or None after assembly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	34 lines (ranges: 33, 49, 52, 55, 58, 60, 63, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 116, 132-133, 180)

AI ASSESSMENT

Scenario: Test the ContextAssembler to assemble a minimal context for a test file.

Why Needed: This test prevents regression where the ContextAssembler is unable to assemble a minimal context for a test file without any additional configuration.

Key Assertions:

- The 'test_1' function should be found in the source code of the test file.
- The 'ContextAssembler' object should have an empty dictionary as its context.
- The 'source' variable should contain the modified source code with the 'test_1' function.
- The 'context' variable should be an empty dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	30 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 116)

AI ASSESSMENT

Scenario: Test the ContextAssembler with balanced context limits to ensure it correctly truncates long content and reports coverage.

Why Needed: This test prevents a potential bug where the ContextAssembler does not truncate long content and instead leaves it in the output.

Key Assertions:

- The 'f1.py' file is present in the assembled context.
- The 'f1.py' file contains the expected truncation message.
- ... The length of the 'f1.py' file is within the allowed limit (20 bytes + truncation message).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	34 lines (ranges: 33, 49, 52, 55, 58, 60-61, 65, 78-79, 82-84, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-156, 158-159, 163, 191-192, 194)

AI ASSESSMENT

Scenario: Test the ContextAssembler to handle non-existent file and nested test names with parameters.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly handles files that do not exist or have nested test names with parameters.

Key Assertions:

- The function `_get_test_source` returns an empty string when given a non-existent file path.
- The function `_get_test_source` correctly extracts the nested test name and parameter from the provided source code.
- The function `_get_test_source` handles nested test names with parameters by including them in the extracted source code.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	26 lines (ranges: 33, 78-79, 82-84, 86-87, 92, 94-95, 98-101, 103-112, 116)

AI ASSESSMENT

Scenario: The test verifies that the ContextAssembler should exclude certain Python files and a secret file from being processed.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly excludes certain files, potentially leading to unexpected behavior or errors.

Key Assertions:

- config_should_exclude('*.pyc') is True
- config_should_exclude('secret/*') is True
- assembler_should_exclude('public/readme.md') is False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	5 lines (ranges: 33, 191-194)

AI ASSESSMENT

Scenario: The 'test_consecutive_lines' test verifies that consecutive lines in a list are compressed using range notation.

Why Needed: This test prevents regression when consecutive lines are not compressed correctly.

Key Assertions:

- assert compress_ranges([1, 2, 3]) == '1-3'
- assert compress_ranges([4, 5, 6]) == '4-6'
- assert compress_ranges([]) == ''
- assert compress_ranges([1]) == '1'
- assert compress_ranges([-1]) == '-1'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

AI ASSESSMENT

Scenario: Test that the function correctly handles duplicate ranges.

Why Needed: Prevents a potential bug where the function incorrectly identifies distinct ranges as duplicates.

Key Assertions:

- The function should return '1-3' for the input range [1, 2, 2, 3, 3, 3].
- The function should not return '1-4' for the input range [1, 2, 2, 3, 3, 4].
- The function should correctly handle ranges with duplicate values in them.
- The function should not incorrectly identify ranges as duplicates when there are no duplicates.
- The function should return '1-5' for the input range [1, 2, 2, 3, 3, 4].
- The function should correctly handle ranges with duplicate values in them (e.g. [1, 2, 2, 3, 3, 3]).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

AI ASSESSMENT

Scenario: Testing the `compress_ranges` function with an empty input list.

Why Needed: This test prevents a potential bug where an empty list is not correctly compressed to a single string.

Key Assertions:

- The function should return an empty string for an empty input list.
- The function should handle the case where `compress_ranges` is called with no arguments (i.e., an empty list) without raising any errors or exceptions.
- The function should produce a correct and meaningful output when given an empty list as input, rather than producing an incorrect or misleading result.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	2 lines (ranges: 29-30)

AI ASSESSMENT

Scenario: Test 'test_mixed_ranges' verifies that the `compress_ranges` function handles mixed ranges correctly.

Why Needed: This test prevents a potential regression where the function incorrectly combines ranges with single values.

Key Assertions:

- The output should be in the format '1-3, 5, 10-12, 15'.
- The range '1' should be included as is.
- The range '2' to '4' should be combined into a single range '2-4'.
- The range '5' to '7' should be combined into a single range '5-7'.
- The range '8' to '10' should be combined into a single range '8-10'.
- The range '11' to '15' should be combined into a single range '11-15'.
- All ranges should have a unique start and end value.
- No single values should be included in the output without a corresponding range.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)

AI ASSESSMENT

Scenario: Test that non-consecutive lines are correctly compressed into a single string.

Why Needed: This test prevents regression in cases where consecutive line numbers are not separated by commas.

Key Assertions:

- The function compresses the input list of integers into a comma-separated string.
- The resulting string contains only the specified line numbers (1, 3, and 5).
- No other line numbers are included in the output string.
- Non-consecutive line numbers are not separated by commas.
- Consecutive line numbers are separated correctly by commas.
- The function handles edge cases where the input list contains only one or two elements.
- It also works correctly when the input list is empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	14 lines (ranges: 29, 33, 35-37, 39-40, 45-47, 50, 52, 65-66)

AI ASSESSMENT

Scenario: tests/test_ranges.py::TestCompressRanges::test_single_line

Why Needed: This test prevents a regression where the single-line function does not correctly handle ranges.

Key Assertions:

- The input list should be empty or contain only one element.
- The compressed string should match the original input.
- No range notation should be used in the output.
- The function should raise an error for invalid inputs (e.g., non-numeric values).
- The function should correctly handle ranges with multiple elements.
- The function should not use any range notation when given a single element.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	10 lines (ranges: 29, 33, 35-37, 39, 50, 52, 65-66)

AI ASSESSMENT

Scenario: tests/test_ranges.py::TestCompressRanges::test_two_consecutive

Why Needed: This test prevents a regression where two consecutive numbers are compressed to a single range.

- Key Assertions:**
- The function should return the correct range notation for two consecutive numbers.
 - The function should handle cases where the input list contains only one number.
 - The function should not compress consecutive ranges of zeros.
 - The function should preserve the original order of non-zero numbers in the input list.
 - The function should handle edge cases such as an empty input list or a list with only one element.
 - The function should return the correct range notation even if the two consecutive numbers are equal.
 - The function should not compress ranges that span multiple lines.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

AI ASSESSMENT

- Scenario:** Test the function with an unsorted list of ranges.
- Why Needed:** This test prevents a potential bug where the function does not handle unsorted input correctly.
- Key Assertions:**
- The output should be in the format '1-3, 5'.
 - The comma-separated values should contain both range keys and values.
 - The ranges should be sorted alphabetically by key.
 - The function should return an empty string if the input list is empty.
 - The function should handle duplicate keys correctly (e.g., '1-2, 3').
 - The function should not raise any exceptions when given unsorted input.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)

AI ASSESSMENT

- Scenario:** Testing the `expand_ranges` function with an empty string.
- Why Needed:** This test prevents a potential bug where the function returns incorrect results for empty strings.
- Key Assertions:**
- The `expand_ranges` function should return an empty list when given an empty string as input.
 - The expected output of the `expand_ranges` function for an empty string is indeed an empty list.
 - The test case verifies that the function handles empty strings correctly and produces no results.
 - Any additional tests or assertions should be added to ensure this specific scenario works as expected.
 - The test should also verify that the function raises a `ValueError` when given invalid input, such as non-string values.
 - To further improve robustness, consider adding error handling for edge cases like empty strings with multiple ranges.
 - A more comprehensive test case could involve checking the function's behavior with different types of inputs, such as lists or tuples.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	2 lines (ranges: 81-82)

AI ASSESSMENT

Scenario: Test 'test_mixed' verifies the expansion of mixed ranges and singles.

Why Needed: This test prevents a potential bug where the expand_ranges function does not correctly handle mixed range values (e.g., '1-3, 5, 10-12')

Key Assertions:

- The expanded list should contain all specified numbers from both ranges and singles.
- Numbers in the first range should be included before those in the second range.
- Single numbers should not be split across multiple ranges.
- Negative numbers should still be treated as single values.
- Zero is a valid number for this test.
- The function should handle cases where the input string contains commas correctly (e.g., '1-3, 5, 10-12').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	11 lines (ranges: 81, 84-91, 93, 95)

AI ASSESSMENT

Scenario: The 'expand_ranges' function is expected to expand a range of numbers.

Why Needed: This test prevents the function from expanding ranges that do not contain all specified numbers.

Key Assertions:

- The input string should be in the format 'start-end'
- The start value should be less than or equal to the end value
- All values in the range should be integers

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	10 lines (ranges: 81, 84-91, 95)

AI ASSESSMENT

Scenario: The test verifies that the `expand_ranges` function correctly reverses the compression of a list.

Why Needed: This test prevents bugs where the inverse operation of `compress_ranges` is not implemented correctly, potentially causing unexpected behavior or errors.

Key Assertions:

- The input list should be unchanged after calling `expand_ranges(compressed)`.
- All elements in the output list should be present in the original list.
- If an element is missing from the original list, it should also be missing from the output list.
- If two or more elements are removed from the original list, they should not be present in the output list.
- The order of elements in the output list should be preserved.
- If all elements are present in both lists (original and expanded), then the compressed list should also be present in the expanded list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	27 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67, 81, 84-91, 93, 95)

AI ASSESSMENT

Scenario: The 'expand_ranges' function should be able to handle a single input, producing the same output as a range of numbers.

Why Needed: This test prevents regression in cases where only one number is provided.

Key Assertions:

- Input: '5'
- Expected Output: [5]
- No error or exception should be raised
- The function should handle the input correctly and produce a list with one element

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	7 lines (ranges: 81, 84-87, 93, 95)

AI ASSESSMENT

Scenario: Test that the function formats duration as milliseconds for less than 1 second.

Why Needed: This test prevents a regression where the function fails to format durations correctly for values greater than or equal to 1 second.

Key Assertions:

- The function should return '500ms' when given a duration of 0.5 seconds.
- The function should return '1ms' when given a duration of 0.001 seconds.
- The function should return '0ms' when given a duration of 0.0 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	2 lines (ranges: 65, 67)

AI ASSESSMENT

Scenario: Test that the function formats seconds correctly.

Why Needed: Prevents a potential bug where seconds are not formatted as expected for values less than 1 second.

Key Assertions:

- The function should return the correct string representation of seconds (e.g. '1.23s') for input values between 0 and 1.
- The function should return the correct string representation of seconds (e.g. '60.00s') for input values greater than or equal to 1 second.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	2 lines (ranges: 65-66)

AI ASSESSMENT

Scenario: Test that all outcomes map to CSS classes correctly.

Why Needed: Prevents regression where 'xfailed' or 'xpassed' are incorrectly mapped to the wrong CSS class.

Key Assertions:

- The function `outcome_to_css_class()` maps 'passed', 'failed', and 'skipped' outcomes to the correct CSS classes ('outcome-passed', 'outcome-failed', 'outcome-skipped')
- The function `outcome_to_css_class('xfailed')` should map to 'outcome-xfailed'
- The function `outcome_to_css_class('xpassed')` should map to 'outcome-xpassed'

COVERAGE

<code>src/pytest_llm_report/collector.py</code>	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
<code>src/pytest_llm_report/plugin.py</code>	6 lines (ranges: 380-381, 384, 388-390)
<code>src/pytest_llm_report/render.py</code>	8 lines (ranges: 79-85, 87)

AI ASSESSMENT

Scenario: tests/test_render.py::TestOutcomeToCssClass::test_unknown_outcome

Why Needed: This test prevents a regression where unknown outcomes are not properly handled and instead default to the 'outcome-unknown' class.

Key Assertions:

- `outcome_to_css_class('unknown') == 'outcome-unknown'`
- `outcome_to_css_class('invalid') != 'outcome-unknown'`
- `outcome_to_css_class('default') == 'outcome-default'`

COVERAGE

<code>src/pytest_llm_report/collector.py</code>	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
<code>src/pytest_llm_report/plugin.py</code>	6 lines (ranges: 380-381, 384, 388-390)
<code>src/pytest_llm_report/render.py</code>	8 lines (ranges: 79-85, 87)

AI ASSESSMENT

Scenario: The test verifies that a basic report is rendered with the expected HTML structure.

Why Needed: This test prevents a rendering issue where the report does not display correctly due to missing or incorrect HTML tags.

Key Assertions:

- The presence of '' in the rendered HTML.
- The presence of 'Test Report' in the rendered HTML.
- The presence of 'test::passed' and 'test::failed' in the rendered HTML.
- The presence of 'PASSED' and 'FAILED' in the rendered HTML.
- The presence of 'Plugin: v0.1.0' and 'Repo: v1.2.3' in the rendered HTML.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	52 lines (ranges: 65-67, 79-85, 87, 121-124, 126-127, 131-132, 141-143, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

AI ASSESSMENT

Scenario: Test renders coverage for fallback HTML.

Why Needed: Prevents regression and ensures accurate coverage reporting.

Key Assertions:

- The test verifies that the `render_fallback_html` function includes the source file `'src/foo.py'` in its rendered HTML.
- The test checks that there are exactly 5 lines of code in the rendered HTML.
- The test verifies that the line ranges for each line range from 1 to 5, indicating coverage.
- The test ensures that the `CoverageEntry` object is created with the correct file path and line count.
- The test checks that the `CoverageEntry` object has a `file_path` attribute matching `'src/foo.py'` and a `line_count` attribute equal to 5.
- The test verifies that the rendered HTML includes all lines of code from the source file, as indicated by the line ranges (1-5).
- The test checks that the rendered HTML does not include any lines outside the specified range (1-5).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	52 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-129, 131-132, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

AI ASSESSMENT

Scenario: Verifies that the report includes LLM annotations for LLMs.

Why Needed: This test prevents authentication bypass by ensuring LLM annotations are included in the report.

Key Assertions:

- The report contains "Tests login flow" as a key assertion.
- The report contains "Prevents auth bypass" as a key assertion.
- The report includes LLM annotations for LLMs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	54 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-127, 131-134, 136-137, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

AI ASSESSMENT

Scenario: Test renders source coverage for fallback HTML.

Why Needed: Prevents regression where missing source code is not properly reported.

Key Assertions:

- The 'Source Coverage' section should be present in the rendered HTML.
- The file path 'src/foo.py' should be included in the 'Source Coverage' section.
- The coverage percentage (80.0%) should be displayed correctly in the HTML.
- The ranges '1-4, 6-8' and '5, 9-10' should be accurately reported as covered or missed.
- The number of statements (10) should be included in the coverage report.
- The number of missed statements (2) should be correctly identified as missing.
- The percentage of covered statements (80.0%) should be calculated and displayed correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	63 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-127, 131-132, 141-142, 145-153, 158-164, 166-172, 177, 192, 196, 229-236, 239-245, 248-249)

AI ASSESSMENT

Scenario: Test 'Should include xfailed/xpassed summary entries' verifies that the rendered report includes XFailed and XPassed summaries.

Why Needed: This test prevents a regression where the summary section is missing or incorrectly formatted.

Key Assertions:

- The string 'XFailed' should be present in the HTML output.
- The string 'XPassed' should be present in the HTML output.
- Both 'XFailed' and 'XPassed' should be included in the rendered report.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	50 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-127, 131-132, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

AI ASSESSMENT

Scenario: Test 'different_content' verifies that the same input produces different hashes.

Why Needed: This test prevents a bug where two inputs with the same content but different origins produce the same hash.

Key Assertions:

- The function `compute_sha256` should return different values for different inputs.
- The output of `compute_sha256(b'hello')` and `compute_sha256(b'world')` should be different.
- The output of `compute_sha256(b'hello') != compute_sha256(b'world')` should be True.
- The hash of `compute_sha256(b'hello')` should not match the hash of `compute_sha256(b'world')`.
- The hash of `compute_sha256(b'hello')` should not be equal to `compute_sha256(b'hello')` (case sensitivity)
- The hash of `compute_sha256(b'hello')` should not be equal to `compute_sha256(b'HELLO')` (case sensitivity)
- The hash of `compute_sha256(b'hello')` should not be equal to `compute_sha256(b'world')` (case insensitivity)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

AI ASSESSMENT

Scenario: Test 'Empty bytes should produce consistent hash' verifies that an empty byte string produces the same hash as a non-empty byte string.

Why Needed: This test prevents a potential bug where the hash of an empty byte string is different from the hash of a non-empty byte string, potentially leading to incorrect reporting or analysis.

Key Assertions:

- The hash of an empty byte string should be equal to the hash of a non-empty byte string (i.e., `hash1 == hash2`).
- The length of the resulting hash should be consistent for both cases (i.e., `len(hash1) == 64`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

AI ASSESSMENT

Scenario: Test builds run metadata with correct version info.

Why Needed: This test prevents regression where the report writer does not include the pytest version in the build run metadata.

Key Assertions:

- The duration of the test should be 60 seconds.
- The pytest version should have a value.
- The plugin version should be '0.1.0'.
- The python version should also be present and correct.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	67 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300)

AI ASSESSMENT

Scenario: Test verifies that the `build_summary` method correctly counts all outcome types and their corresponding values.

Why Needed: This test prevents a potential regression where the count of each outcome type is not accurate due to missing or incorrect data.

Key Assertions:

- The total number of outcomes should be equal to 6 (passed, failed, skipped, xfailed, xpassed, error).
- Each outcome type should have its corresponding value in the `summary` dictionary: passed = 1, failed = 1, skipped = 1, xfailed = 1, xpassed = 1, error = 1.
- The values of each outcome type are correctly assigned to their respective keys in the `summary` dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	19 lines (ranges: 156-158, 312, 314-315, 317-328, 330)

AI ASSESSMENT

Scenario: Test 'test_build_summary_counts' verifies that the summary counts outcomes correctly.

Why Needed: This test prevents a bug where the summary incorrectly counts failed tests as passed.

Key Assertions:

- asserts that the total count of all tests is equal to 4
- asserts that the number of passed tests is equal to 2
- asserts that the number of failed tests is equal to 1
- asserts that the number of skipped tests is equal to 1

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	13 lines (ranges: 156-158, 312, 314-315, 317-322, 330)

AI ASSESSMENT

Scenario: Test that the Writer initializes correctly with a given configuration.

Why Needed: This test prevents a potential bug where the Writer does not properly initialize with the provided configuration, potentially leading to incorrect or missing data in reports.

Key Assertions:

- The ``config`` attribute of the ``writer`` object is set to the specified ``Config`` instance.
- The ``warnings`` list of the ``writer`` object is empty.
- The ``artifacts`` list of the ``writer`` object is empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	3 lines (ranges: 156-158)

AI ASSESSMENT

Scenario: Test that ReportWriter writes a report with all tests.

Why Needed: This test prevents regression where the report does not include all tests, potentially causing confusion or missing important information.

Key Assertions:

- The length of the report.tests list should be equal to 2.
- The value of report.summary.total should be equal to 2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	93 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330)

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class writes a report with a total coverage percentage.

Why Needed: This test prevents regression where the coverage percentage is not included in the report.

Key Assertions:

- The `report.summary.coverage_total_percent` attribute should be equal to the provided `coverage_percent` value.
- The `report.summary.coverage_total_percent` attribute should contain only numeric values (e.g., integers or floats).
- The `report.summary.coverage_total_percent` attribute should not exceed 100% if coverage is above 100%
- The `report.summary.coverage_total_percent` attribute should be calculated correctly based on the provided `coverage_percent` value.
- The `report.summary.coverage_total_percent` attribute should be updated after writing a new report with different `coverage_percent` values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	93 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-199, 202-206, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330)

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_includes_source_coverage verifies that the test writes a report with includes source coverage.

Why Needed: This test prevents regression where the report does not include source coverage, potentially misleading users about the code's quality.

Key Assertions:

- The length of the `source_coverage` list in the report should be exactly 1.
- The file path of the first `SourceCoverageEntry` in the `source_coverage` list should match 'src/foo.py'.
- All statements covered by the source code should be included in the coverage summary.
- At least one statement from the missed code should be included in the coverage summary.
- All covered lines should have a percentage greater than or equal to 87.5%.
- The `covered_ranges` attribute of each `SourceCoverageEntry` should match '1-4, 6-7'.
- At least one line from the missed code should be included in the coverage summary.
- All covered lines should have a percentage greater than or equal to 87.5% and less than 100%.
- The `missed_ranges` attribute of each `SourceCoverageEntry` should match '5'.
- The total number of statements, missed, and covered lines in the coverage summary should be consistent across all test runs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	92 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330)

AI ASSESSMENT

Scenario: Report should merge coverage into tests.

Why Needed: This test prevents a bug where the report does not correctly merge coverage from multiple tests, leading to incorrect reporting of test coverage.

Key Assertions:

- `assert len(report.tests[0].coverage) == 1`
- `assert report.tests[0].coverage[0].file_path == 'src/foo.py'`
- `assert all(isinstance(c, CoverageEntry) for c in report.tests[0].coverage)`
- `assert all('file_path' in c.__dict__ for c in report.tests[0].coverage)`
- `assert len(report.tests[0].coverage['test1']) == 1`
- `assert report.tests[0].coverage['test1'][0] is not None`
- `assert isinstance(report.tests[0].coverage['test1'][0], CoverageEntry)`
- `assert 'file_path' in report.tests[0].coverage['test1'][0].__dict__`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	94 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186-189, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330)

AI ASSESSMENT

Scenario: Test the fallback to direct write when atomic write fails.

Why Needed: This test prevents a regression where an atomic write operation fails and the direct write is used instead, potentially leading to unexpected behavior or data loss.

Key Assertions:

- Verify that the report.json file exists at the expected path.
- Verify that any warnings have code 'W203' when the direct write fails.
- Verify that the direct write has been performed successfully by checking for the existence of the report.json file.
- Verify that all warnings in the direct write are of type 'W203'.
- Verify that there are no other warnings or errors in the writer's output.
- Verify that the direct write does not fail and return an error code.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	67 lines (ranges: 229-231, 233, 235, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	125 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506-507, 509-512, 515-516)

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` creates an output directory if it doesn't exist.

Why Needed: This test prevents a bug where the report writer fails to create a directory when the input JSON file does not exist.

Key Assertions:

- If `tmp_path / 'subdir' / 'report.json'` does not exist, then `tmp_path / 'subdir' / 'report.json'.exists()` should return `True`.
- The output directory created by the report writer should have the correct name (`'subdir'`) and be located in the specified path (`tmp_path.subdir`).
- If a test case is passed, then `tmp_path / 'subdir' / 'report.json'.exists()` should return `False`.
- The output directory created by the report writer should have the correct permissions (i.e., read and write access for the current user).
- The output directory created by the report writer should be a valid JSON file with the expected structure.
- If an error occurs while writing the report, then `tmp_path / 'subdir' / 'report.json'.exists()` should return `False`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	84 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 229-231, 233, 235, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	123 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315,

317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-477, 495, 497, 499-501, 503, 506)

PASSED

tests/test_report_writer.py::TestReportWriterWithFiles::test_ensure_dir_failure

1ms  4

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class raises a warning when attempting to create a directory that already exists.

Why Needed: This test prevents a potential bug where the `ReportWriter` class does not raise an error when trying to write a report in a non-existent directory.

Key Assertions:

- The function `writer._ensure_dir(json_path)` should raise a `FileExistsError` with code 'W201' when creating a directory that already exists.
- The `writer.warnings` list should contain at least one warning object with code 'W201' when attempting to create the non-existent directory.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	12 lines (ranges: 156-158, 470-473, 480-484)

AI ASSESSMENT

Scenario: Test the report writer to handle git command failures gracefully.

Why Needed: This test prevents a regression where the report writer fails to retrieve git information when it's not found.

Key Assertions:

- The function `get_git_info()` should return `None` for both SHA and dirty flag if git is not found.
- The function `get_git_info()` should raise an exception with message 'Git not found' if git command fails.
- The test should pass even when the subprocess call to check_output raises an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	9 lines (ranges: 67-73, 85-86)

AI ASSESSMENT

Scenario: Test verifies that the `write_report` method creates an HTML file with expected content.

Why Needed: This test prevents a regression where the report writer does not create an HTML file, potentially leading to missing or incorrect report data.

Key Assertions:

- The file should exist at the specified path.
- The HTML file should contain the expected content (test1, test2, PASSED, FAILED, Skipped, XFailed, XPassed, Errors).
- All nodes in the report should be present in the HTML file.
- Each node type (PASSED, FAILED, Skipped, XFailed, XPassed) should be included in the HTML content.
- The 'Errors' and 'XPassed/XFailed' node types should be present in the HTML file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	115 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Test 'test_write_html_includes_xfail_summary' verifies that the report writer includes xfail outcomes in the HTML summary.

Why Needed: This test prevents a bug where the report does not include xfail outcomes in the HTML summary, potentially misleading users about the status of tests.

Key Assertions:

- The 'XFAILED' and 'XPASSED' keywords are present in the HTML summary.
- The 'xfailed' and 'xpassed' keywords are present in the HTML summary.
- All xfail outcomes are included in the HTML summary.
- No xpass outcomes are included in the HTML summary.
- The report does not include any unknown or unreported test results.
- The report includes all expected test results, including xfail and xpassed outcomes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	118 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321, 323-326, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Test verifies that a JSON file is created with the report.

Why Needed: This test prevents regression where the report writer does not create a JSON file.

Key Assertions:

- The `report.json` file should exist at the specified path.
- At least one artifact should be tracked in the JSON file.
- The length of the artifacts list should be greater than zero.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	78 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	117 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Should create PDF file when Playwright is available.

Why Needed: This test prevents regression that would occur if the playwright module was not available or could not be imported correctly.

Key Assertions:

- The `write_pdf` function from the `ReportWriter` class should successfully write a PDF file to the specified path.
- The `report.pdf` attribute of the `writer` object should contain the expected file path.
- Any artifacts created by the report writer should have the correct path relative to the `report.pdf` file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	125 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230-231, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 401, 410, 412, 414-423, 434-435, 437-443, 448, 453, 455, 458-462, 470-471)

AI ASSESSMENT

Scenario: Test that a warning is raised when PDF output is requested without Playwright.

Why Needed: To prevent the test from failing due to a missing required module (Playwright).

Key Assertions:

- The file 'report.pdf' should not exist.
- Any warnings with code `WarningCode.W204_PDF_PLAYWRIGHT_MISSING` value should be present.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	98 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230-231, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 401-405, 408)

AI ASSESSMENT

Scenario: Ensures directory creation of report writer output files.

Why Needed: Prevents a potential issue where the report writer does not create the required directory structure for HTML files.

Key Assertions:

- The `tmp_dir / 'r.html'` path exists before any warnings are printed.
- Any warning messages (code 'W202') are present in the output file.
- The `tmp_dir / 'r.html'` path is deleted after writing the report.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	11 lines (ranges: 156-158, 470-477)

AI ASSESSMENT

Scenario: Tests the scenario where report_writer_metadata_skips verifies that metadata skips when reports are disabled.

Why Needed: This test prevents regression by ensuring that metadata is skipped when reports are disabled, which is a critical check for accurate reporting.

Key Assertions:

- The 'start_time' key should be present in the metadata.
- Metadata should not contain an 'llm_model' key.
- The 'llm_model' value should be None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	36 lines (ranges: 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	67 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300)

AI ASSESSMENT

Scenario: Test that `AnnotationSchema.from_dict` can create a valid annotation from a dictionary with all required fields.

Why Needed: Prevents regression where the `from_dict` method is used incorrectly, potentially causing invalid annotations to be created.

Key Assertions:

- `assert schema.scenario == 'Verify login'`
- `assert schema.why_needed == 'Catch auth bugs'`
- `assert schema.key_assertions == ['assert 200', 'assert token']`
- `assert schema.confidence == 0.95`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test converting AnnotationSchema to dictionary with all fields.

Why Needed: Prevents regression in schema conversion logic, ensuring accurate representation of annotation data.

- Key Assertions:**
- assert 'scenario' in data and data['scenario'] == 'Verify login'
 - assert 'why_needed' in data and data['why_needed'] == 'Catch auth bugs'
 - assert 'key_assertions' in data and data['key_assertions'].all()
 - assert 200 in data['key_assertions'] and data['key_assertions'][0] == 'assert 200'
 - assert token in data['key_assertions'] and data['key_assertions'][1] == 'assert token'
 - assert confidence in data and data['confidence'] == 0.95

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	8 lines (ranges: 90-92, 94-98)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: The HTML report is generated correctly and can be accessed.

Why Needed: This test prevents a regression where the report might not be created or accessible due to changes in the pytester's environment.

Key Assertions:

- The file path of the report should exist after running the test.
- The content of the report should contain " and 'test_simple' as expected.
- The function name 'test_simple' should be present in the report content.

COVERAGE

src/pytest_llm_report/collector.py	39 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)

src/pytest_llm_report/render.py

25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)

src/pytest_llm_report/report_writer.py

101 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: test_html_summary_counts_all_statuses verifies that all statuses are included in the HTML summary.

Why Needed: This test prevents regression where the count of all statuses is missing from the report.

Key Assertions:

- asserts that 'Total Tests' and 'Errors' labels appear in the report
- asserts that 'Passed', 'Failed', 'Skipped', 'XFailed', and 'XPassed' labels appear in the report
- asserts that 'Errors' label appears only once in the report

COVERAGE

src/pytest_llm_report/collector.py	65 lines (ranges: 78-79, 90, 93-94, 96, 99-104, 106-107, 109-112, 114-119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212-214, 216, 227-228, 230-236, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390,

401, 405, 424, 428-430, 441, 445, 448, 450-451)

src/pytest_llm_report/render.py

25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)

src/pytest_llm_report/report_writer.py

111 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-328, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: The JSON report is created successfully.

Why Needed: This test prevents a potential bug where the report generation process fails to create the expected JSON file.

Key Assertions:

- The `report_path` exists after running the test.
- The `data` dictionary in the `report_path` contains the correct schema version and summary statistics.
- The total number of tests passed is equal to the total number of tests failed.
- At least one test passed and at least one test failed.

COVERAGE

src/pytest_llm_report/collector.py	51 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-118, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 227-228, 230-236, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151,

153-156, 169-171, 173-175,
177-179, 183, 187-188, 190,
192, 195-196, 203, 212-213,
238, 242, 246, 249, 268-269,
276-277, 280-281, 283-284,
287-291, 293, 296-297, 299,
302-303, 324, 330-331, 358-
368, 380-381, 384, 388-390,
401, 405, 424, 428-430, 441,
445, 448, 450-451)

src/pytest_llm_report/report_writer.py

107 lines (ranges: 55, 67-73,
85-86, 98-100, 127-128, 130,
156-158, 186, 192-193, 197-
198, 202, 211-218, 222-223,
226, 230, 233, 254, 256-259,
262-264, 266, 268-275, 277-
278, 280-289, 291-294, 296-
297, 299-300, 312, 314-315,
317-320, 330, 340, 343-345,
348-349, 352-354, 357, 360-
364, 470-471, 495, 497, 499-
501, 503, 506)

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report generated by pytester for a provider enabled.

Why Needed: Prevents regressions and ensures that LLM annotations are properly included in the report.

Key Assertions:

- The scenario 'Checks the happy path' is present in the report.
- The reason 'Prevents regressions' is present in the report.
- The key assertions 'asserts True' are present in the report.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	39 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198, 203)
src/pytest_llm_report/llm/base.py	39 lines (ranges: 52-53, 72, 75, 80, 107, 110-111, 128, 136, 147, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218, 245, 247, 249, 252, 257-258, 260)
src/pytest_llm_report/llm/litellm_provider.py	23 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 66-67, 69-70, 94-95, 97)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)

src/pytest_llm_report/models.py	94 lines (ranges: 104-107, 109-111, 113, 115, 161-165, 167, 169, 171, 173, 176, 178-180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407-419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	47 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	186 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-333, 336, 338, 341-345, 348, 350-355, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
src/pytest_llm_report/prompts.py	29 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-109, 111-112, 116)
src/pytest_llm_report/report_writer.py	105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-296, 298-299, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Verifies that LLM errors are surfaced in HTML output.

Why Needed: Prevents regression where LLM errors are not reported correctly.

Key Assertions:

- The test verifies the presence of 'LLM error' and 'boom' in the report content.
- The test asserts that both 'LLM error' and 'boom' are found in the report content.
- The test checks for the correct spelling of 'LLM error' and 'boom'.
- The test verifies that the LLM error is reported correctly, not just raised.
- The test ensures that the error message contains both 'LLM error' and 'boom'.

COVERAGE

src/pytest_llm_report/cache.py	12 lines (ranges: 39-41, 53, 55-56, 86, 88, 118-119, 121, 153)
src/pytest_llm_report/collector.py	39 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/llm/annotator.py	73 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137-139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198-201, 203)
src/pytest_llm_report/llm/base.py	21 lines (ranges: 52-53, 72, 75, 80, 107, 110-111, 128, 136, 147, 165, 167, 175, 245, 247, 249, 252, 257-258, 260)
src/pytest_llm_report/llm/litellm_provider.py	25 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 78-79, 81-82, 84-85, 94-95, 97)
src/pytest_llm_report/options.py	47 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-

259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)

src/pytest_llm_report/plugin.py

186 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-333, 336, 338, 341-346, 350-355, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)

src/pytest_llm_report/prompts.py

29 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-109, 111-112, 116)

src/pytest_llm_report/render.py

25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)

src/pytest_llm_report/report_writer.py

101 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-296, 298-299, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Test the LLM opt-out marker functionality.

Why Needed: Prevents regression in LLM opt-out marker detection, ensuring that all test cases are correctly marked as 'opted out'.

Key Assertions:

- The function `test_opt_out()` is called with the correct arguments.
- The `llm_opt_out` attribute of each test case is set to `True` after running the LLM opt-out marker.
- The number of tests that pass (i.e., are marked as 'opted out') remains unchanged across different runs.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181-182, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180-182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190,

192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)

src/pytest_llm_report/report_writer.py

105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: The test verifies that a requirement marker is correctly recorded and verified.

Why Needed: This test prevents a potential bug where the requirement marker might not be recorded or verified correctly, potentially leading to false positives or negatives in the test results.

Key Assertions:

- The `pytest.mark.requirement` decorator is applied to the `test_with_req` function with two requirements: 'REQ-001' and 'REQ-002'.
- The `pytester.runpytest` command is used to run the tests with a custom report file, which includes the requirement markers.
- The test data is loaded from the report file using `json.loads`, and the length of the tests list is verified to be 1.
- The requirements for each test are extracted from the test data and verified to contain 'REQ-001' and 'REQ-002'.
- The `pytester.path` attribute is used to get the path to the report file, which is a required argument for the `runpytest` command.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-200, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188-190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)

src/pytest_llm_report/plugin.py

166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)

src/pytest_llm_report/report_writer.py

105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Test 'Multiple xfailed tests are recorded in the report' verifies that multiple xfailed tests are correctly reported.

Why Needed: This test prevents regression by ensuring that multiple xfailed tests are not missed or incorrectly counted.

Key Assertions:

- The total number of xfailed tests is equal to the sum of individual test outcomes.
- Each xfailed test has a corresponding outcome in the report (xfailed and xpassed).
- Multiple xfailed tests are correctly recorded in the report, without any duplicates or omissions.

COVERAGE

src/pytest_llm_report/collector.py	47 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-116, 119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212, 216, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151,

153-156, 169-171, 173-175,
177-179, 183, 187-188, 190,
192, 195-196, 203, 212-213,
238, 242, 246, 249, 268-269,
276-277, 280-281, 283-284,
287-291, 293, 296-297, 299,
302-303, 324, 330-331, 358-
368, 380-381, 384, 388-390,
401, 405, 424, 428-430, 441,
445, 448, 450-451)

src/pytest_llm_report/report_writer.py

108 lines (ranges: 55, 67-73,
85-86, 98-100, 127-128, 130,
156-158, 186, 192-193, 197-
198, 202, 211-218, 222-223,
226, 230, 233, 254, 256-259,
262-264, 266, 268-275, 277-
278, 280-289, 291-294, 296-
297, 299-300, 312, 314-315,
317, 319, 321, 323-324, 330,
340, 343-345, 348-349, 352-
354, 357, 360-364, 470-471,
495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Test that skipped tests are recorded and their count is verified.

Why Needed: This test prevents a regression where the number of skipped tests is not correctly reported.

Key Assertions:

- The 'summary' key in the report.json file should contain the correct number of skipped tests.
- The value of the 'skipped' key in the 'summary' dictionary should be equal to 1.
- The total count of skipped tests should match the actual number of skipped tests in the test suite.

COVERAGE

src/pytest_llm_report/collector.py	43 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 106-107, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175,

177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)

src/pytest_llm_report/report_writer.py

107 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321-322, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Verifies that the test 'test_xfail' is marked as Xfailed and its outcome is recorded in the report.

Why Needed: This test prevents a regression where the 'test_xfail' function is not properly marked as Xfailed, causing it to be counted towards the total number of failed tests.

Key Assertions:

- The 'test_xfail' function is marked with the @pytest.mark.xfail marker.
- The outcome of the test 'test_xfail' is recorded in the report.
- The value of 'xfailed' in the report matches the number of times the 'test_xfail' function was run.
- The total count of failed tests is updated correctly after running the test.
- The 'summary' section of the report includes the correct key-value pair for 'xfailed'.

COVERAGE

src/pytest_llm_report/collector.py	47 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-116, 119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212, 216, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117,

121-124, 126-129, 131-134,
136-140, 142-145, 147-151,
153-156, 169-171, 173-175,
177-179, 183, 187-188, 190,
192, 195-196, 203, 212-213,
238, 242, 246, 249, 268-269,
276-277, 280-281, 283-284,
287-291, 293, 296-297, 299,
302-303, 324, 330-331, 358-
368, 380-381, 384, 388-390,
401, 405, 424, 428-430, 441,
445, 448, 450-451)

src/pytest_llm_report/report_writer.py

108 lines (ranges: 55, 67-73,
85-86, 98-100, 127-128, 130,
156-158, 186, 192-193, 197-
198, 202, 211-218, 222-223,
226, 230, 233, 254, 256-259,
262-264, 266, 268-275, 277-
278, 280-289, 291-294, 296-
297, 299-300, 312, 314-315,
317, 319, 321, 323-324, 330,
340, 343-345, 348-349, 352-
354, 357, 360-364, 470-471,
495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: Test the parameterized tests feature to ensure it records and runs correctly.

Why Needed: This test prevents regression by verifying that the parameterized tests are recorded separately and run with the correct report.

Key Assertions:

- The total number of tests in the report is 3 (1 passed, 2 failed).
- Each test has a unique name (test_param) and a valid assertion (assert x > 0).

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163-164, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167, 169-171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284,

287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)

src/pytest_llm_report/report_writer.py

105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

AI ASSESSMENT

Scenario: The CLI help text should include usage examples.

Why Needed: This test prevents a potential bug where the help message does not contain any usage examples, making it difficult for users to understand how to use the plugin.

Key Assertions:

- assert result.stdout.fnmatch_lines(['*Example:*--llm-report*'])
- assert 'Example:' in result.stdout
- assert '--llm-report' in result.stdout

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	45 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	118 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that LLM markers are registered and correctly displayed in the pytest output.

Why Needed: This test prevents a bug where LLM markers are not properly registered or displayed, potentially causing issues with the test suite.

Key Assertions:

- The 'llm_opt_out' marker should be found in the stdout of the pytest run.
- The 'llm_context' marker should be found in the stdout of the pytest run.
- The 'requirement' marker should be found in the stdout of the pytest run.
- The 'llm_opt_out', 'llm_context', and 'requirement' markers should match the expected lines in the stdout.
- The 'llm_opt_out' marker should not be matched with any other marker in the stdout.
- The 'llm_context' marker should not be matched with any other marker in the stdout.
- The 'requirement' marker should not be matched with any other marker in the stdout.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	45 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	118 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that the plugin is correctly registered via pytest11 and displays the LLM report in stdout.

Why Needed: This test prevents a potential issue where the plugin might not be properly registered or configured, potentially leading to incorrect results or errors during testing.

Key Assertions:

- The `pytester.runpytest` command is executed with the `--help` option.
- The output of the `stdout` stream contains the string 'LLM report'.
- The plugin is registered correctly via pytest11.
- The LLM report is displayed in stdout.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	45 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	118 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 380-381, 384, 388-390)

AI ASSESSMENT

Scenario: Test that special characters in nodeid are handled correctly by pytester.

Why Needed: This test prevents a potential crash and ensures the HTML generated is valid.

Key Assertions:

- The `report.html` file should exist after running the test.
- The `report.html` file should contain `''` in its content.
- The nodeid parameter in pytester.makepyfile does not cause a crash but instead generates an HTML report with valid content.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163-164, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)

src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	101 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED

tests/test_time.py::TestFormatDuration::test_boundary_one_minute

1ms



AI ASSESSMENT

Scenario: Tests the 'format_duration' function with a boundary of exactly one minute.

Why Needed: This test prevents a potential bug where the function incorrectly formats minutes as seconds instead of seconds.

Key Assertions:

- The result should be in the format '1m 0.0s'.
- The result should contain only one unit (either 's' for seconds or 'm' for minutes).
- The value of '0.0s' should not exceed the maximum allowed length.
- The function should handle cases where the input is less than or equal to zero.
- The function should handle cases where the input is exactly one minute (60 seconds).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

AI ASSESSMENT

Scenario: Tests the `format_duration` function to ensure it correctly formats sub-millisecond durations as microseconds.

Why Needed: This test prevents a potential bug where the function does not format durations with microseconds when they are less than one millisecond.

Key Assertions:

- The result of calling `format_duration(0.0005)` should contain the string `'μs'` (microseconds).
- The result of calling `format_duration(0.001)` should be equal to `'1μs'`.
- The function should correctly format durations with microseconds, even when they are less than one millisecond.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	2 lines (ranges: 39-40)

AI ASSESSMENT

Scenario: Verifies that the `format_duration` function correctly formats sub-second durations as milliseconds.

Why Needed: Prevents a potential bug where the format string does not include 'ms' for all sub-second durations.

Key Assertions:

- The result of calling `format_duration(0.5)` should contain the string 'ms'.
- The value of `result` should be equal to '500.0ms'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	3 lines (ranges: 39, 41-42)

AI ASSESSMENT

Scenario: Test the `format_duration` function to verify it correctly formats durations over a minute.

Why Needed: This test prevents regression where the function incorrectly returns 'm' instead of 'mm' for minutes.

Key Assertions:

- The function should return '1m 30.5s' after formatting a duration of 90.5 seconds.
- The assertion should check if the returned string contains the character 'm'.
- The assertion should compare the result with the expected string '1m 30.5s'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a scenario that formats multiple minutes.

Why Needed: This test prevents regression in cases where the input duration is in minutes, as it may be incorrectly formatted to seconds.

Key Assertions:

- The result of calling `format_duration(185.0)` should be `'3m 5.0s'`.
- The format string should include a unit of `'m'` for minutes and an optional unit of `'s'` for seconds.
- The function should correctly handle cases where the input duration is in minutes but not in seconds (e.g., `185.25`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of exactly one second.

Why Needed: Prevents regression in time-related functionality where a single-second duration is expected.

Key Assertions:

- The function correctly formats the duration as '1.00s'.
- The function returns an error if the input duration is not exactly one second.
- The function does not silently fail for non-numeric input durations.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

AI ASSESSMENT

Scenario: Test the 'seconds' format for durations under one minute.

Why Needed: Prevents regression where seconds are not correctly formatted as 'xx.s' (e.g., 10.0s becomes 10.00s).

Key Assertions:

- The result contains the string 's', indicating it is in seconds format.
- The result equals '5.50s' to ensure correct conversion.
- The result does not contain any non-numeric characters, ensuring it is a valid time string.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 1 millisecond.

Why Needed: Prevents regression in formatting small durations to milliseconds.

Key Assertions:

- The output should be '1.0ms' when the input is 1 millisecond.
- The unit should be 'ms' (milli-seconds).
- No decimal places should be displayed for the duration value.
- The function should correctly handle durations in the range of milliseconds.
- No exceptions should be raised if the input is not a non-negative number.
- The output should match the expected string representation when converted to a float.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	3 lines (ranges: 39, 41-42)

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a very small duration (1 microsecond).

Why Needed: This test prevents a potential bug where the function incorrectly formats durations less than 1 millisecond.

Key Assertions:

- The output of `format_duration(0.000001)` should be `'1μs'`.
- The value of `result` is equal to `'1μs'` when passed as an argument to `format_duration`.
- The function correctly formats durations less than 1 millisecond.
- The function handles negative values and zero correctly.
- The function does not throw any exceptions for invalid input (e.g., non-numeric values).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	2 lines (ranges: 39-40)

AI ASSESSMENT

Scenario: Test the functionality of datetime with UTC timezone in ISO format.

Why Needed: This test prevents a potential bug where datetime objects from other timezones are not correctly formatted into UTC timezone.

Key Assertions:

- The datetime object is created with the correct timezone (UTC).
- The `iso_format` function returns the expected result (`'2024-01-15T10:30:45+00:00'`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

AI ASSESSMENT

Scenario: Verifies that naive datetime formats are correctly converted to ISO format without timezone.

Why Needed: Prevents a potential bug where naive datetime formats may not be correctly converted to ISO format, potentially leading to incorrect results or errors in downstream applications.

Key Assertions:

- The function `iso_format(dt)` correctly converts the input `dt` (2024-06-20T14:00:00) to an ISO formatted string "2024-06-20T14:00:00".
- The function does not add any timezone information to the output.
- The function handles edge cases where the input datetime is in a format that is not supported by `iso_format` (e.g. invalid date or time values).
- The function raises an error if the input datetime is not a valid ISO formatted string.
- The function preserves the original timezone information of the input datetime.
- The function correctly handles different cases where the input datetime is in UTC, EST, etc.
- The function does not silently ignore or suppress errors that occur during conversion (e.g. invalid date/time values).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

AI ASSESSMENT

Scenario: Test the `iso_format` function with datetime objects that include microseconds.

Why Needed: This test prevents a potential issue where the `iso_format` function may not correctly format dates with microseconds if the input datetime object does not have enough time components.

Key Assertions:

- The result of calling `iso_format(dt)` should contain the string '123456'.
- The result of calling `iso_format(dt)` should include the substring '123456' in its value.
- The format string passed to `iso_format(dt)` should be able to correctly handle microseconds by including them in the output.
- The `datetime` object passed to `iso_format(dt)` should have a time component that includes microseconds.
- The `tzinfo` parameter of the `datetime` object passed to `iso_format(dt)` is set to UTC.
- The `result` variable should contain the string '123456' after calling `iso_format(dt)`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

AI ASSESSMENT

Scenario: Verifies that the ``utc_now()`` function returns a datetime object with an associated UTC timezone.

Why Needed: Prevents regression in tests where the test environment does not have a valid UTC timezone.

Key Assertions:

- `result.tzinfo` is not `None`
- `result.tzinfo == UTC`
- `assert result.tzinfo is not None and result.tzinfo == UTC`

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)

AI ASSESSMENT

Scenario: Verifies that the ``utc_now()`` function returns a time within UTC.

Why Needed: Prevents regression where the current time is not correctly identified as being in UTC.

Key Assertions:

- The ``before`` variable should be less than or equal to the ``result`` and greater than or equal to the ``after`` variables.
- The ``result`` should be within a tolerance of ``before`` and ``after``.
- The ``utc_now()`` function correctly identifies the current time as being in UTC.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)

AI ASSESSMENT

Scenario: The `utc_now()` function should return a datetime object.

Why Needed: This test prevents a potential bug where the function returns an incorrect type of value (datetime vs. datetime.datetime).

Key Assertions:

- result is an instance of datetime or datetime.datetime
- result is not None
- result has a valid timezone
- result is not a string
- result is not a timedelta
- result is not a date
- result is not a time

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)

Source Coverage

FILE	STMTS	MISS	COVER	%	COVERED LINES	MISSED LINES
src/pytest_llm_report/_git_info.py	2	0	2	100.0%	2-3	-
src/pytest_llm_report/aggregate.py	116	5	111	95.69%	13, 15-19, 21, 35, 38, 44, 46, 52-53, 55-57, 59, 61-64, 69, 73-74, 77-80, 84, 87-89, 93, 103, 109-111, 113-117, 119-120, 125, 127-128, 130-131, 134-135, 141-144, 146, 148, 162, 164, 168, 170, 172, 182, 184-188, 190-191, 194,	66, 90-91, 192, 203

						196, 205, 217, 219-233, 235, 237, 245-246, 248-249, 251, 253-255, 259, 262-263, 265-266, 269, 271-272, 274, 276-277, 281
--	--	--	--	--	--	---

src/pytest_llm_report/cache.py	47	3	44	93.62%	13, 15-19, 21, 27, 33, 39-41, 43, 53, 55-56, 58, 60-62, 68-69, 78, 86, 88, 90, 92, 94, 97, 103, 107, 118-119, 121, 123, 129, 132-136, 141, 144, 153	64-65, 130
--------------------------------	----	---	----	--------	--	------------

src/pytest_llm_report/collector.py	111	2	109	98.2%	19, 21-22, 24, 26-27, 33-34, 45- 50, 52, 58, 60- 62, 69, 78-79, 81, 90, 93-94, 96, 99-104, 106- 107, 109-112, 114-119, 121-122, 124, 127-128, 130, 132-133, 135-137, 140, 143, 155, 163- 164, 167-169, 171, 173, 181- 182, 185-189, 191, 198-200, 202, 209-210, 212-214, 216, 218, 227-228, 230-236, 238, 241, 250-252, 254, 261, 264- 265, 268-269, 271, 277, 279, 285	141, 239
------------------------------------	-----	---	-----	-------	---	----------

--	--	--	--	--	--	--

src/pytest_llm_report/coverage_map.py	135	10	125	92.59%	13, 15-17, 19-22, 30, 38, 44-45, 47, 58-60, 64, 72-73, 83, 86, 88-90, 92, 94-96, 98, 101-104, 106-108, 114, 116, 118, 121-122, 127, 131-135, 137-140, 144-146, 148, 150, 152-153, 156, 160-162, 165, 167-168, 173, 176, 178-184, 187-189, 191, 196, 199-200, 202, 204, 216-217, 220, 224-225, 228-234, 236, 239, 241, 243-244, 246-248, 250, 252-254, 259-260, 263-264, 271, 273, 276-279, 281-283, 285, 299-300, 302, 308	62, 123, 125, 128, 157, 221, 249, 251, 257, 274
src/pytest_llm_report/errors.py	35	0	35	100.0%	8-9, 12, 25-28, 31-36, 39-42, 45-46, 49-51, 54-55, 64-66, 68, 70, 74-76, 80, 129, 139	-
src/pytest_llm_report/llm/__init__.py	3	0	3	100.0%	4-5, 7	-

src/pytest_llm_report/llm/annotator.py

110

0

110

100.0%

4, 6-10, 12-15, 21-22, 25-28, 31, 45-46, 48-50, 54, 56-57, 59, 61-62, 64, 66-68, 71-72, 74-82, 87, 97-98, 100, 102, 104-105, 115, 127, 129-132, 137-139, 142, 165-168, 170-171, 176, 178, 180-183, 185-190, 192-193, 198-201, 203, 206, 229-232, 234, 236-237, 239-240, 245-246, 248-253, 255-256, 261-264, 266

src/pytest_llm_report/llm/base.py

78

0

78

100.0%

13, 15-18, 26, 40, 46, 52-53, 55, 72, 75-76, 78, 80, 101, 107-108, 110-111, 122, 128, 130, 136, 138, 147, 149, 165, 167-173, 175, 177, 186-187, 190-192, 194-195, 198-200, 203-208, 212, 214, 220-221, 224-225, 228-230, 233, 245, 247, 249-250, 252-253, 255, 257-258, 260, 262-263, 265, 267

src/pytest_llm_report/llm/gemini.py	275	18	257	93.45%	7, 9-13, 15-16, 23-27, 30-34, 37-42, 44-46, 48-50, 52, 57-63, 65-70, 72-73, 75-78, 80-85, 87-88, 91-97, 99-103, 105, 107-114, 121-122, 125, 128, 134, 136-139, 141-142, 144, 160-161, 167-169, 171-172, 174, 176-184, 186-188, 190-191, 193, 196, 200-208, 210-211, 213-215, 217-223, 225-227, 233-234, 238-239, 242-243, 245-248, 252-253, 260, 266-267, 269, 273-277, 279-283, 286-287, 292-293, 300-301, 303, 315, 317-318, 322, 327, 330-332, 335-343, 345-346, 348, 352-355, 357, 360-366, 368-374, 380-382, 384-387, 389, 391-392, 396-402, 405, 408-410, 412-414, 416-421, 427-428, 430-434, 437-440, 442-443, 445-447
					89, 104, 106, 115-117, 199, 230-231, 235-237, 244, 250, 256, 367, 441, 444
src/pytest_llm_report/llm/litellm_provider.py	32	1	31	96.88%	7, 9, 11-12, 18, 21, 37-38, 44, 46, 49, 51-52, 54-56, 66-67, 69-70, 73, 76, 78-79, 81-82, 84, 88, 94-95, 97
src/pytest_llm_report/llm/noop.py	13	0	13	100.0%	8, 10, 12-13, 20, 26, 32, 34, 50, 52, 58, 60, 66
src/pytest_llm_report/llm/ollama.py	43	1	42	97.67%	7, 9, 11-12, 18, 24, 40-41, 47, 50, 52, 54-55, 57-60, 62-63, 66-67, 71-72, 74-75, 77, 81, 87-88, 90-92, 96, 102, 69

src/pytest_llm_report/plugin.py	151	24	127	84.11%	40, 43, 49, 55, 61, 67, 73, 80, 89, 95, 101, 107, 113, 121, 126, 131, 136, 142, 147, 153, 169, 173, 177, 183-184, 187-188, 190, 192, 195-197, 203-204, 212-213, 238-239, 242-243, 246, 249-250, 252-253, 256-257, 259, 261-265, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-305, 307, 309-312, 315-316, 324-325, 330-333, 336, 338, 341-346, 348, 350, 358-359, 380-381, 384-385, 388-390, 401-402, 405, 408-409, 412-414, 424-425, 428-430, 441-442, 445, 448, 450-451
					13, 15-17, 19-20, 22, 28-31, 34, 160, 216, 320-321, 326-327, 372-373, 393, 417, 433-434
src/pytest_llm_report/prompts.py	75	5	70	93.33%	13, 15-17, 24, 27, 33, 35, 49, 52, 55, 58-61, 63, 65, 67, 78-79, 82-84, 86-87, 92, 94-95, 98-101, 103-112, 116, 118, 132-133, 135-138, 140-141, 144-145, 148, 151-152, 154-156, 158-159, 163, 165, 180, 182, 191-194
src/pytest_llm_report/reporter.py	50	0	50	100.0%	13, 15-16, 18, 24, 30-31, 34, 40, 42, 50-51, 53, 56, 65-67, 70, 79, 87, 90, 99, 101-102, 107, 110, 121-124, 126-129, 131-134, -

					141-143, 145, 158-163, 177, 196
					13, 15-25, 27-29, 46, 55, 58, 67- 68, 76, 83-84, 89, 98-100, 102, 105-108, 110, 116, 127-128, 130, 142, 150, 156-158, 160, 186-189, 192, 197-199, 202-203, 211, 222-223, 226-227, 230-231, 233, 235, 254, 256-259, 262-264, 266, 268, 303, 312, 314-315, 317-328, 330, 332, 340, 343- 345, 348-349, 352-354, 357, 360, 368, 376, 378-379, 382, 385, 388, 391, 399, 401-402, 408, 410, 412, 414-423, 434-435, 437-439, 447-448, 453, 455, 458, 461-462, 464, 470-474, 480-481, 488, 495, 497, 499-501, 503, 506-507, 509, 515-516
src/pytest_llm_report/re port_writer.py	167	10	157	94.01%	113, 135-137, 424-425, 432, 449-451
					11, 13-14, 17, 30, 33, 36, 39, 42, 45, 55-56, 58-60, 63-64, 70, 79, 82, 100, 103, 111-113, 116-117, 119-121, 123
src/pytest_llm_report/ut il/fs.py	34	3	31	91.18%	40, 65, 67
					12, 14-17, 23, 32, 35, 44-48, 51, 61, 64, 73- 74, 76-78, 80-81, - 86, 96, 103-104, 107, 113-114, 116-121
src/pytest_llm_report/ut il/hashing.py	36	0	36	100.0%	

src/pytest_llm_report/ut	33	0	33	100.0%	12, 15, 29-30, 33, 35-37, 39-40, 42, 45-47, 50, 52, 55, 65-67, 70, 81-82, 84-91, 93, 95	-
il/ranges.py						

src/pytest_llm_report/ut	16	0	16	100.0%	4, 6, 9, 15, 18, 27, 30, 39-44, 46-48	-
il/time.py						