Test Report

Run ID: 20971419458-py3.12 • Generated: 2026-01-13 20:27:06 • Duration: 34.69s
Plugin: v0.1.0 (2f498263985a34902252c53c11fb820445bd8f21) [dirty]
Repo: v0.1.0 (c88e8ba83476730ae529aa8303ba0bf384e527ba)
LLM: ollama / llama3.2:1b (minimal context, 379 annotated, 7 errors)

92.91%
Total Coverage

387 TOTAL TESTS	387 PASSED	0 FAILED	0 SKIPPED
0 XFAILED	0 XPASSED	0 ERRORS	

PASSED `tests/test_aggregation.py::TestAggregator::test_aggregate_all_policy` 2ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'test_aggregate_all_policy' verifies that the aggregate function correctly aggregates all policy tests.

Why Needed: This test prevents regression where a single policy test fails to aggregate with other tests, potentially leading to incorrect results or false positives.

Key Assertions:

- The aggregated report should contain both retained policy tests.
- The number of retained policy tests in the aggregated report should be equal to 2.
- Each retained policy test should have a unique 'nodeid' and 'outcome' key in the aggregated report.
- The aggregated report should not be empty.
- All retained policy tests should have a duration greater than 0.
- All retained policy tests should have a phase of 'call'.
- The aggregate function should correctly handle duplicate nodeids and outcomes by retaining only one instance of each.
- The aggregate function should preserve the original order of policy tests in the input reports.

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, 275-278, 280)
src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_aggregation.py::TestAggregator::test_aggregate_dir_not_exists 4ms 🛡️ 3

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 work correctly if the input directory does not exist.

Key Assertions:

- The `aggregate` method of the `aggregator` object should return `None` when the input directory does not exist.
- The `aggregate` method of the `aggregator` object should raise an error or handle the case correctly if the input directory does not exist.
- The test should fail when the input directory exists, indicating a bug in the aggregate function.
- The test should pass when the input directory does not exist, indicating that the aggregate function is working as expected.
- The `exists` method of the `Path` object should return `False` for the input directory.
- The `exists` method of the `Path` object should raise an error if the input path is invalid or does not exist.
- The `aggregate` method of the `aggregator` object should handle the case where the input directory exists correctly.

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: 387-388, 391, 395-397)

PASSED `tests/test_aggregation.py::TestAggregator::test_aggregate_latest_policy` 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the test_aggregate_latest_policy function correctly selects the latest policy for aggregation.

Why Needed: This test prevents regression where the test case is run multiple times in a row, causing the test to always pick the first report's latest policy.

Key Assertions:

- The test verifies that the `aggregate` method returns the correct latest policy for each test case.
- The test checks that the returned policy from `aggregate` is indeed the latest for each test case.
- The test ensures that the aggregated run meta contains the correct number of tests and their respective outcomes.
- The test verifies that the summary indicates a passed count for the latest policy.
- The test asserts that the aggregate result has no failed tests.
- The test checks that the `run_meta` object is set to indicate an aggregated run.
- The test verifies that the `summary` attribute of the `run_meta` object correctly reflects the outcome of each test case.

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, 275-278, 280)
src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_aggregation.py::TestAggregator::test_aggregate_no_dir_configured` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that aggregate function returns None when no directory configuration is provided.

Why Needed: Prevents a potential bug where the aggregate function would raise an error due to missing directory configuration.

Key Assertions:

- The `aggregate()` method of the `Aggregator` class should return `None` when called with a mock configuration that does not specify an aggregation directory.
- The `aggregate_dir` attribute of the `Aggregator` instance should be set to `None` after calling `aggregate()`.
- An error message or exception should not be raised when calling `aggregate()` with a mock configuration that does not specify an aggregation directory.
- The `aggregate()` method should behave as expected without raising any exceptions or errors.
- The `aggregate_dir` attribute of the `Aggregator` instance should remain unchanged after calling `aggregate()`.
- The test should be able to reproduce the issue consistently across different environments and configurations.

COVERAGE

src/pytest_llm_report/aggregation.py	3 lines (ranges: 44, 52-53)
src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_aggregation.py::TestAggregator::test_aggregate_no_reports` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The `aggregate` method of the Aggregator class should not be called when there are no reports.

Why Needed: This test prevents a potential bug where the aggregate method might be called with an empty list or set of reports, potentially causing unexpected behavior or errors.

Key Assertions:

- aggregator.aggregate() is None
- pathlib.Path.exists() returns True
- pathlib.Path.glob() returns []

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: 387-388, 391, 395-397)

PASSED `tests/test_aggregation.py::TestAggregator::test_aggregate_with_coverage_and_llm_annotations` 2ms 🛡️ 4

AI ASSESSMENT

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

Why Needed: Prevents regression in core functionality by ensuring accurate coverage and LLM annotation deserialization.

Key Assertions:

- Coverage was correctly deserialized from the JSON report.
- LLM annotation was correctly deserialized from the JSON report.
- The aggregated result can be re-serialized without any issues.
- The serialized report contains the expected keys (coverage and LLM annotation).
- The coverage ranges match the original file paths.
- The line counts in the coverage entries match the original file lines.
- The confidence level of the LLM annotation matches the original value.
- The key assertions were correctly extracted from the aggregated result.

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, 275-278, 280)
src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_aggregation.py::TestAggregator::test_aggregate_with_source_coverage` 2ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that source coverage summary is deserialized correctly when aggregated with a report.

Why Needed: This test prevents regression where the source coverage summary is not properly deserialized when aggregating reports from different directories.

Key Assertions:

- The `source_coverage` attribute of the result object should be an instance of `SourceCoverageEntry`.
- The `file_path` attribute of the first element in the `source_coverage` list should match the expected file path.
- All elements in the `source_coverage` list should have a valid `coverage_percent`, `covered_ranges`, and `missed_ranges` attribute.
- Each `SourceCoverageEntry` object should have a `file_path` attribute that matches the expected file path.
- The `covered_ranges` attribute should be a string containing two ranges separated by a comma (e.g., '1-5, 7-11').
- The `missed_ranges` attribute should be an empty string.
- All statements in the source code should be present in the aggregated report.
- The coverage percentage should be greater than or equal to 0% and less than or equal to 100%
- The number of covered statements should be greater than or equal to the number of missed statements
- The total coverage percentage should be calculated correctly based on the source code coverage

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, 275-278, 280)
src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_aggregation.py::TestAggregator::test_load_coverage_from_source` 3ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: Prevents regression where coverage data is missing due to lack of configuration.

Key Assertions:

- Verify that the _load_coverage_from_source method returns None when llm_coverage_source is set to None.
- Verify that the _load_coverage_from_source method raises a UserWarning with message 'Coverage source not found' when llm_coverage_source is set to '/nonexistent/coverage'.
- Verify that the _load_coverage_from_source method correctly loads coverage data from a mock .coverage file when llm_coverage_source is set to '.coverage'.

COVERAGE

`src/pytest_llm_report/aggregation.py`	19 lines (ranges: 245-246, 248-249, 251, 253-257, 259, 262-263, 265-266, 269-271, 273)
`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_aggregation.py::TestAggregator::test_recalculate_summary` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: test_recalculate_summary verifies that the _recalculate_summary method preserves coverage when aggregating test results.

Why Needed: This test prevents regression in the aggregation of test results, ensuring that the coverage percentage is preserved even after multiple calls to _recalculate_summary.

Key Assertions:

- summary.total == 6 (total number of tests)
- summary.passed == 1 (number of passed tests)
- summary.failed == 1 (number of failed tests)
- summary.skipped == 1 (number of skipped tests)
- summary.xfailed == 1 (number of xfailed tests)
- summary.xpassed == 1 (number of xpassed tests)
- summary.error == 1 (number of error tests)
- summary.coverage_total_percent == 85.5 (coverage percentage preserved)
- summary.total_duration == 5.0 (total duration of all tests)

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: 387-388, 391, 395-397)

PASSED `tests/test_aggregation.py::TestAggregator::test_skips_invalid_json` 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Test Skips Invalid JSON: Verifies that the test prevents skipping of reports with non-JSON content.

Why Needed: This test prevents skipping of reports containing invalid or missing fields in their JSON format, ensuring that all reports are processed correctly.

Key Assertions:

- The `aggregate` function should not be called on a report with a 'not json' file.
- The `run_meta.run_count` attribute should still return the correct count even if only one valid report is found.
- The test should fail when skipping invalid reports, indicating that the test logic is correct.

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, 275-278, 280)
src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_aggregation_maximal.py::TestAggregationMaximal::test_recalculate_summary_coverage` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the aggregator recalculates the summary correctly when there are multiple tests with different outcomes.

Why Needed: This test prevents regression in the aggregation process, ensuring that the coverage total percent is calculated accurately even if some tests fail.

Key Assertions:

- summary.total == 2
- summary.passed == 1
- summary.failed == 1
- summary.coverage_total_percent == 88.5
- summary.total_duration == 3.0

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: 387-388, 391, 395-397)`

PASSED tests/test_annotator.py::TestAnnotateTests::test_cached_tests_are_skipped 2ms 🛡️ 5

AI ASSESSMENT

Scenario: Verify that cached tests are skipped by the annotator.

Why Needed: This test prevents regression in cases where the annotator is caching test results and should skip them to avoid re-running unnecessary tests.

Key Assertions:

- The mock provider, cache, and assembler objects are created with `mock_provider`, `mock_cache`, and `mock_assembler` respectively.
- The `test_cached_tests_are_skipped` method is called with the expected arguments.
- The `test_cached_tests_are_skipped` method checks if the cached test results should be skipped based on the provided conditions.
- The `test_cached_tests_are_skipped` method returns a boolean indicating whether the cached tests are skipped or not.
- The `test_cached_tests_are_skipped` method asserts that the return value is as expected.
- The `test_cached_tests_are_skipped` method checks if the annotator should skip the test based on the provided conditions.
- The `test_cached_tests_are_skipped` method uses the mock objects to simulate the caching process and the annotation logic.

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: 387-388, 391, 395-397)

PASSED tests/test_annotator.py::TestAnnotateTests::test_concurrent_annotation 3ms 🛡️ 5

AI ASSESSMENT

Scenario: The `test_concurrent_annotation` function is being tested to ensure that annotators can process multiple annotations concurrently without any issues.

Why Needed: This test prevents potential performance regressions or bugs caused by concurrent annotation processing.

Key Assertions:

- The `mock_provider`, `mock_cache`, and `mock_assembler` objects are not modified during the execution of this function.
- The annotator's output is not affected by concurrent annotations.
- The test function does not throw any exceptions or errors when running concurrently with other tests.
- The annotator's performance remains stable even under concurrent annotation processing.
- The cache size and annotation storage are not modified during the execution of this function.
- The `mock_provider` object is updated correctly after each annotation is processed.
- No unexpected side effects occur on the test environment when running concurrently with other tests.

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: 387-388, 391, 395-397)

PASSED `tests/test_annotator.py::TestAnnotateTests::test_concurrent_annotation_handles_failures` 2ms 🛡️ 5

AI ASSESSMENT

Scenario: Test concurrent annotation handles failures to verify

Why Needed: This test prevents a bug where annotators fail to handle failures in the annotation process.

Key Assertions:

- Mocking `mock_provider` and `mock_assembler` with mock objects that raise an exception when called concurrently is necessary to ensure the annotations are handled correctly.
- The `mock_cache` object should not be affected by concurrent calls to `test_concurrent_annotation_handles_failures`.
- When `mock_provider` raises an exception, it should be caught and handled by the test, preventing failures from propagating through the annotation process.
- Similarly, when `mock_assembler` raises an exception, it should also be caught and handled correctly by the test.
- The `mock_cache` object should not store any data that would prevent subsequent calls to `test_concurrent_annotation_handles_failures` from succeeding.
- If a failure occurs in the annotation process, the test should still be able to verify that the annotator handles it correctly and continues processing other tasks.
- In case of concurrent failures, the test should not fail due to one task being unable to complete before another task completes its work.
- The test should also ensure that any exceptions raised by `mock_provider` or `mock_assembler` are properly propagated up the call stack.

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: 387-388, 391, 395-397)

PASSED tests/test_annotator.py::TestAnnotateTests::test_progress_reporting 2ms 🛡️ 5

AI ASSESSMENT

Scenario: The `test_progress_reporting` function is being tested to ensure it correctly reports progress for annotators.

Why Needed: This test prevents regressions that may occur if the progress reporting functionality changes without updating the key assertions.

Key Assertions:

- Verify that the progress reporter updates with correct keys.
- Check if the progress reporter increments the `annotator_id` and `total_annotations` keys correctly.
- Verify that the progress reporter resets the `annotator_id` and `total_annotations` keys when a new annotator is added.
- Confirm that the progress reporter displays the correct number of annotations for each annotator.
- Check if the progress reporter updates the `progress` key with the correct percentage.
- Verify that the progress reporter increments the `annotations_completed` key correctly after each annotation.
- Confirm that the progress reporter resets the `annotations_completed` key when a new annotation is added.
- Check if the progress reporter displays the correct number of annotations for each annotator across all annotators.

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: 387-388, 391, 395-397)

PASSED `tests/test_annotator.py::TestAnnotateTests::test_sequential_annotation` 12.00s 🛡️ 5

AI ASSESSMENT

Scenario: Verifies the sequential annotation functionality of the annotator.

Why Needed: Prevents regression in sequential annotation logic when using multiple annotators.

Key Assertions:

- The function should correctly annotate text sequentially by passing a list of annotations to the `annotate` method.
- The function should handle cases where the input list is empty or contains only one annotation.
- The function should not throw an exception if the input list is None.
- The function should return the correct number of annotated tokens for each sentence.
- The function should correctly handle case-insensitive tokenization and stemming.
- The function should support annotators with different tokenization and stemming algorithms.
- The function should be able to handle complex sentences with multiple clauses.
- The function should not throw an exception if the input list contains annotations that are not valid for sequential annotation.

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: 387-388, 391, 395-397)

PASSED `tests/test_annotator.py::TestAnnotateTests::test_skips_if_disabled` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The `test_skips_if_disabled` test verifies that annotating tests with a disabled LLM configuration does not have any effect.

Why Needed: This test prevents regression in the case where the LLM is disabled, ensuring that annotating tests still works as expected.

Key Assertions:

- The `test_skips_if_disabled` function should not attempt to annotate any tests when the LLM is disabled.
- The `annotate_tests` function should not be called with an empty list of annotations when the LLM is disabled.
- The test should still pass without any errors or exceptions even though the LLM is disabled.
- The configuration object passed to `annotate_tests` should have a 'LLM' key set to False.
- The annotation process should not attempt to skip tests that are annotated with an LLM-enabled configuration.
- The test should be able to annotate tests without skipping them even though the LLM is disabled.
- The `test_skips_if_disabled` function should not have any side effects on the test result.
- The test should still report a successful outcome for all annotated tests.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_annotator.py::TestAnnotateTests::test_skips_if_provider_unavailable` 1ms 🛡️ 4

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the annotator fails to skip annotations due to an unavailable provider.

Key Assertions:

- Mocking `mock_provider` with an available mock instance should allow the annotator to skip the annotation process.
- The `capsys` fixture is not being used in this test, which could prevent it from capturing the skipped annotation message.
- The `mock_provider` should be mocked to return a successful response (200 OK) when the provider is unavailable.
- The annotator should still be able to skip annotations even if the provider returns an error (4xx or 5xx status code).
- The test should not fail due to the availability of the provider, but rather because it's unavailable.
- The annotator's behavior should remain consistent with previous tests where providers were available.
- The `mock_provider` instance should be properly cleaned up after the test completes.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_annotator_maximal.py::TestAnnotatorAdvanced::test_annotate_concurrent_with_progress_and_errors` 2ms 🛡️ 4

AI ASSESSMENT

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

Why Needed: This test prevents regression where the annotator fails to report progress or the first error when annotated concurrently with progress and errors.

Key Assertions:

- The function should append a message indicating processing of the second task.
- The function should append a message indicating that an LLM annotation occurred for the second task.
- The function should return at least two annotations (one success and one error).
- The function should report 'first error' in the first error message.
- The function should report 'Processing 2 test(s)' in some progress messages.
- The function should report 'LLM annotation' in some progress 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/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: 387-388, 391, 395-397)

PASSED `tests/test_annotator_maximal.py::TestAnnotatorAdvanced::test_annotate_sequential_rate_limit_wait` 2ms 🛡️ 4

AI ASSESSMENT

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

Why Needed: This test prevents regression in cases where the annotator takes longer than expected to complete tasks due to a slow rate limit.

Key Assertions:

- assert mock_sleep.called
- assert mock_time.call_count == 5
- assert mock_time.side_effect == [100.0, 100.1, 100.2, 100.3, 100.4]
- assert tasks[-1].outcome != 'done' because of the slow rate limit interval

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_annotator_maximal.py::TestAnnotatorAdvanced::test_annotate_tests_cached_progress 2ms 🛡️ 5

AI ASSESSMENT

Scenario: Test verifies that the annotator reports cached tests' progress correctly.

Why Needed: This test prevents regression where the annotator does not report progress for cached tests.

Key Assertions:

- The 'cache': test_cached message is present in the progress messages.
- The 'cache': test_cached message contains the scenario 'cached'.
- The 'cache': test_cached message indicates that the annotation was successful.
- The 'src' key in the progress messages corresponds to the source of the cached annotations.
- The 'None' value under the 'src' key in the progress messages is present for all tests.
- The annotator returns a progress message with the correct scenario ('cached') for each test.
- The annotator does not return any progress messages for non-cached tests.

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: 387-388, 391, 395-397)

PASSED `tests/test_annotator_maximal.py::TestAnnotatorAdvanced::test_annotate_tests_provider_unavailable` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that the annotator fails when the test provider is not available.

Why Needed: To prevent regression and ensure the annotator behaves as expected when the test provider is unavailable.

Key Assertions:

- mocks.is_available.assert_called_once_with('ollama')
- annotate_tests.mock.get_provider().is_available.called_once_with('ollama')
- mocks.is_available.return_value False
- assert captured.out contains 'not available. Skipping annotations'
- mocks.get_provider().get_provider() returns None
- assert mock_provider.is_available.call_count == 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/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: 387-388, 391, 395-397)

PASSED tests/test_base_coverage_v2.py::test_base_parse_response_malformed_json_after_extract 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that extracting a malformed JSON from an LLM response will result in a `JSONDecodeError`.

Why Needed: This test prevents the regression of a potential bug where extracting a malformed JSON from an LLM response would not raise a `JSONDecodeError`, but instead return a valid JSON object.

Key Assertions:

- The extracted JSON should be invalid and contain braces.
- The extracted JSON should have a `json_decode()` method that raises a `JSONDecodeError` with the message 'Failed to parse LLM response as JSON'.
- The error message should include information about the malformed JSON, such as its contents.
- The test should fail when the extracted JSON is valid but contains invalid braces.
- The test should pass when the extracted JSON is invalid and does not contain any valid JSON syntax.
- The test should pass even if the `json_decode()` method raises a different exception than `JSONDecodeError`.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_base_coverage_v2.py::test_base_parse_response_non_string_fields` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Verify that the `MockProvider` correctly parses a response with non-string fields and identifies the required key assertions.

Why Needed: Prevents regression in parsing responses with non-string fields, ensuring correct identification of expected keys.

Key Assertions:

- The value of the 'scenario' field should be an integer.
- The list 'why_needed' should contain the string 'list'.
- The key assertion 'a' should exist in the annotation.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_base_maximal.py::TestGetProvider::test_get_gemini_provider` 1ms 🛡️ 5

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where a non-Gemini provider is returned instead of the correct one.

Key Assertions:

- The function `get_provider(config)` should return an instance of `GeminiProvider`
- The function `get_provider(config)` should raise an exception if it cannot find a matching provider
- The function `get_provider(config)` should not return a non-Gemini provider instance

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_base_maximal.py::TestGetProvider::test_get_invalid_provider 2ms 🛡️ 4

AI ASSESSMENT

Scenario: Testing the `get_invalid_provider` method with an invalid provider.

Why Needed: This test prevents a potential bug where an unknown LLM provider is attempted to be used.

Key Assertions:

- The `get_provider` function should raise a `ValueError` when given an invalid provider.
- The error message should indicate that the provided provider is unknown.
- The `pytest.raises` context manager should be able to detect and report the exception.
- The test should fail with the specified error message when running it.
- The `match` parameter of the `pytest.raises` context manager should match the expected error message.
- The `Config` class's `provider` attribute should be set to an invalid value.
- The `get_provider` function should not raise a `ValueError` with an unknown 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/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: 387-388, 391, 395-397)

PASSED tests/test_base_maximal.py::TestGetProvider::test_get_litellm_provider 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Verify that the `get_litellm_provider` function returns a valid instance of `LiteLLMProvider`.

Why Needed: This test prevents a potential bug where the `get_litellm_provider` function might return an incorrect or null provider.

Key Assertions:

- The returned provider should be an instance of `LiteLLMProvider`.
- The provider should not be null or None.
- The provider should have the correct type (`LiteLLMProvider`) as its class.
- The provider's attributes (e.g., `model_name`, `device_id`) should match the expected values.
- The provider's methods (e.g., `get_model`, `set_device`) should be available and functional.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_base_maximal.py::TestGetProvider::test_get_noop_provider` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Verify that the `get_noop_provider` function returns a NoopProvider instance when no provider is specified.

Why Needed: This test prevents a potential bug where the `get_provider` function returns an incorrect type of provider (e.g., a non-None provider) when no provider is provided.

Key Assertions:

- The `provider` attribute of the returned `NoopProvider` instance is `None`.
- The `type()` method of the returned `NoopProvider` instance returns `NoopProvider`.
- The `__class__` attribute of the returned `NoopProvider` instance is `NoopProvider`.
- The `get_provider()` function call does not raise an exception.
- The `provider` variable is assigned a value that is not `None`.
- The `provider` variable is assigned a value that is not an instance of `NoopProvider`.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_base_maximal.py::TestGetProvider::test_get_ollama_provider 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Verifies that the `get_ollama_provider` function returns an instance of OllamaProvider.

Why Needed: Prevents a potential bug where the test fails if the provider is not set to 'ollama'.

Key Assertions:

- The returned value should be an instance of OllamaProvider.
- The returned value should have the correct class name (OllamaProvider).
- The returned value should be a valid provider instance.
- The `get_provider` function is correctly configured with the 'ollama' provider.
- The configuration object passed to `Config` has the required keys ('provider').
- The provider string passed to `Config` is not empty.
- The provider string passed to `Config` starts with '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, 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: 387-388, 391, 395-397)

PASSED `tests/test_base_maximal.py::TestLlmProviderDefaults::test_available_caches_result` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Verify that the LlmProvider is available and has a single check.

Why Needed: This test prevents regression in case of multiple providers or large configurations.

Key Assertions:

- The `is_available()` method returns True for both instances of the provider.
- The `checks` attribute of the provider instance is set to 1 after calling `_check_availability()`.
- Multiple calls to `_check_availability()` will not increase the `checks` counter.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_base_maximal.py::TestLlmProviderDefaults::test_get_model_name_defaults_to_config` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Ensures that the `get_model_name` method returns the default model name from the configuration when no custom model is specified.

Why Needed: This test prevents a regression where the `get_model_name` method does not return the expected default model name when no custom model is provided.

Key Assertions:

- The `provider.get_model_name()` call should return 'test-model'.
- The `provider.get_model_name()` call should be able to determine the default model name from the configuration without any custom model being specified.
- The `provider.get_model_name()` call should not throw an exception when no custom model is 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	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: 387-388, 391, 395-397)

PASSED `tests/test_base_maximal.py::TestLlmProviderDefaults::test_get_rate_limits_defaults_to_none` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Verifies that the default rate limits for LLM providers are set to None when no configuration is provided.

Why Needed: This test prevents a potential bug where the default rate limits are not properly initialized with None, potentially causing unexpected behavior in downstream applications.

Key Assertions:

- The `get_rate_limits()` method of the `ConcreteProvider` class returns `None` when no configuration is provided.
- The `rate_limits` attribute of the provider instance does not have a valid default value (i.e., it's not set to None).
- The `rate_limits` attribute of the provider instance has a valid default value (i.e., it's set to a list or dictionary with at least one element), but this value is not correctly initialized.
- The `get_rate_limits()` method does not raise an exception when no configuration is provided, which could lead to unexpected behavior in downstream 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/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: 387-388, 391, 395-397)`

PASSED tests/test_base_maximal.py::TestLlmProviderDefaults::test_is_local_defaults_to_false 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Verifies that the `is_local()` method returns False for a non-local configuration.

Why Needed: Prevents regression in case of incorrect or outdated configuration settings.

Key Assertions:

- The `provider.is_local()` method is called with an instance of `Config`.
- The `provider.is_local()` method checks if the `local` setting is set to True.
- The `is_local()` method returns a boolean value indicating whether the provider is local or not.
- The test asserts that the returned value is False for non-local configurations.
- The test verifies that the assertion passes when the configuration is correct but outdated.
- The test ensures that the assertion fails when the configuration is incorrect or outdated.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED `tests/test_cache.py::TestHashSource::test_consistent_hash` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the hash of a function with the same source code is equal to its own hash.

Why Needed: This ensures consistency in cache behavior and prevents unexpected collisions between different versions of the same source code.

Key Assertions:

- source == source
- hash_source(source) == hash_source(source)
- source.__code__.co_filename == source.__code__.co_filename

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: 387-388, 391, 395-397)

PASSED tests/test_cache.py::TestHashSource::test_different_source_different_hash 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Testing the behavior of different sources with the same function.

Why Needed: This test prevents a potential bug where two functions with the same name but different source code produce the same hash value.

Key Assertions:

- The function `hash_source` should return a different hash value for two different source strings.
- The function `hash_source` should not raise an error when given the same input.
- The function `hash_source` should correctly handle the case where the input is a string containing multiple words or phrases.
- The function `hash_source` should be able to distinguish between functions with similar names but different source code.
- The function `hash_source` should not produce the same hash value for two different functions with the same name and source code.
- The function `hash_source` should raise an error when given invalid input, such as a non-string or non-function value.

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: 387-388, 391, 395-397)

PASSED `tests/test_cache.py::TestHashSource::test_hash_length` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies the length of the hash generated by the `hash_source` function.

Why Needed: Prevents a potential issue where the hash is not exactly 16 characters long, which could lead to unexpected behavior in certain applications.

Key Assertions:

- The length of the hash should be exactly 16 characters.
- The hash should have no leading zeros (e.g., `0x12345678`).
- No whitespace characters should be present in the hash.
- No special characters or non-ASCII characters should be present in the hash.
- The hash should not be empty.

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: 387-388, 391, 395-397)

PASSED `tests/test_cache.py::TestLlmCache::test_clear` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that clearing the cache removes all entries.

Why Needed: Prevents a regression where adding multiple annotations to the cache and then clearing it would leave some entries behind.

Key Assertions:

- The cache should be cleared with a count of 2.
- The annotation 'test::a' should be removed from the cache with a value of None.
- The annotation 'test::b' should be removed from the cache with a value of None.

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: 387-388, 391, 395-397)

PASSED `tests/test_cache.py::TestLlmCache::test_does_not_cache_errors` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that annotations with errors do not get cached.

Why Needed: To prevent caching of error-related annotations, which could lead to incorrect results or data loss in production environments.

Key Assertions:

- The annotation 'error' is set for the key 'test::foo'.
- The value of the annotation 'error' is 'API timeout'.
- The cache does not store the annotation with the given key and error message.
- If the annotation is retrieved from the cache, it should return None.

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: 387-388, 391, 395-397)`

PASSED `tests/test_cache.py::TestLlmCache::test_get_missing` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test case 'test_get_missing' verifies that the `get` method returns `None` for missing entries in the cache.

Why Needed: This test prevents a potential bug where the `get` method returns `None` when an entry is not found in the cache, causing unexpected behavior or errors.

Key Assertions:

- The `cache.get()` method should return `None` for a non-existent key.
- The `result` variable should be set to `None` after calling `cache.get()`.
- The test should fail when an entry is not found in the cache, indicating that the bug is present.
- The `assert` statement should raise an AssertionError with a meaningful message if the expected behavior is not met.
- The `result` variable should be set to `None` before calling `assert result is None`.
- The test should fail when an entry is not found in the cache, indicating that the bug is present.
- The `cache.get()` method should raise a KeyError with a meaningful message if the key does not exist in the cache.
- The `config.cache_dir` attribute should be set to a valid path for the cache directory.

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: 387-388, 391, 395-397)

PASSED `tests/test_cache.py::TestLlmCache::test_set_and_get` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the functionality of setting and retrieving annotations in the LLMCache.

Why Needed: Prevents bypass attacks by ensuring that cache contents are not tampered with.

Key Assertions:

- Verify that the annotation is correctly stored in the cache.
- Check if the retrieved annotation matches the expected scenario and confidence level.
- Ensure that the cache does not return an empty result when a valid annotation is found.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector.py::TestCollectorCollectionErrors::test_collection_error_structure` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Tests the structure of collection errors to ensure they match their expected format.

Why Needed: To prevent a potential bug where incorrect or missing information is reported in collection errors.

Key Assertions:

- The error nodeid should be set to the name of the file that caused the error.
- The error message should contain the actual error message.
- The error message should not be empty.
- The error message should only contain the syntax-related information (e.g., 'SyntaxError').
- The nodeid should match the expected value provided in the test.
- The message should not contain any additional information that is not relevant to the error.
- The message should not be longer than 50 characters.
- The message should only contain alphanumeric characters and underscores.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_collector.py::TestCollectorCollectionErrors::test_get_collection_errors_initially_empty` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that an empty collection is returned when the `get_collection_errors` method is called on a newly created `TestCollector` instance with an empty configuration.

Why Needed: This test prevents a potential regression where an empty collection might be returned unexpectedly without any errors.

Key Assertions:

- The `get_collection_errors()` method should return an empty list when the input collection is empty.
- An error message or exception should not be raised if the input collection is empty.
- The test should fail with a meaningful error message when the input collection is empty, indicating that it's expected to be empty.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector.py::TestCollectorMarkerExtraction::test_llm_context_override_default_none` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Verifies that the default llm_context_override is set to None for a TestCaseResult.

Why Needed: This test prevents a bug where the default llm_context_override is not correctly set to None for certain cases.

Key Assertions:

- The llm_context_override attribute of TestCaseResult is indeed None.
- If llm_context_override is not None, it should be set to None.
- The default value of llm_context_override is None 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_collector.py::TestCollectorMarkerExtraction::test_llm_opt_out_default_false` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that the default LLM opt-out value is correctly set to False.

Why Needed: Prevents a regression where the default LLM opt-out value might be incorrectly set to True.

Key Assertions:

- The llm_opt_out attribute of TestCaseResult nodeid='test.py::test_foo' should be False.
- The llm_opt_out attribute of TestCaseResult nodeid='test.py::test_foo' is not equal to 'True'.
- The llm_opt_out attribute of TestCaseResult nodeid='test.py::test_foo' is a boolean value (False or 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: 387-388, 391, 395-397)

PASSED `tests/test_collector.py::TestCollectorOutputCapture::test_capture_disabled_by_default` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the default behavior of capturing failed outputs might not be as expected.

Key Assertions:

- config.capture_failed_output should be set to False
- the captured output should not contain any error messages
- no exception should be raised when calling capture() with no arguments
- the captured output should not have a 'capture' attribute
- the captured output should not have an 'error' attribute
- the captured output should not have a 'message' attribute

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_collector.py::TestCollectorOutputCapture::test_capture_max_chars_default` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the default value of `capture_output_max_chars` in the `Config` class is 4000.

Why Needed: This test prevents a potential bug where the maximum characters to capture is not set to a reasonable default value (in this case, 4000).

Key Assertions:

- assert config.capture_output_max_chars == 4000
- assert isinstance(config.capture_output_max_chars, int)
- config.capture_output_max_chars should be greater than or equal to 1
- config.capture_output_max_chars should not exceed 10000
- config.capture_output_max_chars is a positive integer
- config.capture_output_max_chars is an integer value
- config.capture_output_max_chars 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`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_collector.py::TestCollectorXfailHandling::test_xfail_failed_is_xfailed` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that xfail failures are correctly recorded as xfailed in the TestCollector.

Why Needed: This test prevents regression where xfail failures are not properly recorded as expected failure.

Key Assertions:

- The `wasxfail` key in the report should be set to 'expected failure'.
- The `outcome` field of the result object should be set to 'xfailed'.
- The `nodeid` field of the report should match the expected node id.
- The `when` field of the report should match the expected when condition.
- The `passed` field of the report should be False, indicating that the test failed.
- The `skipped` field of the report should be False, indicating that the test was not skipped.
- The `duration` field of the report should be a small value (e.g. 0.01 seconds).
- The `longrepr` field of the report should be an AssertionError message.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector.py::TestCollectorXfailHandling::test_xfail_passed_is_xpassed` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test verifies that xfail passes are recorded as xpassed.

Why Needed: This test prevents regression where xfail passes are not correctly recorded as xpassed.

Key Assertions:

- The `results` dictionary of the collector contains a key-value pair with 'outcome' set to 'xpassed'.
- The value of `result.outcome` is equal to 'xpassed'.
- The `nodeid` in the `results` dictionary matches the expected node id 'test_xfail.py::test_unexpected_pass'.
- The `when` field in the `results` dictionary is set to 'call', which indicates a test run.
- The `duration` and `longrepr` fields are both empty strings, indicating no issues with these metrics.
- The `wasxfail` field matches the expected value '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: 387-388, 391, 395-397)

PASSED `tests/test_collector.py::TestTestCollector::test_create_collector` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the collector does not initialize with empty results.

Key Assertions:

- The `results` attribute of the `collector` object is set to an empty dictionary.
- The `collection_errors` list is empty.
- The `collected_count` attribute of the `collector` object is set to 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/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_collector.py::TestTestCollector::test_get_results_sorted` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `get_results` method returns a list of node IDs sorted by their values.

Why Needed: This test prevents regression where the order of results is not preserved due to manual sorting.

Key Assertions:

- The output list should contain the same nodes as the input list, but in ascending order.
- The `nodeid` attribute of each result object should be present and match the expected values.
- No duplicate node IDs should be included in the output list.
- All node IDs should be sorted alphabetically (case-insensitive).
- The sorting is done correctly even if there are multiple results with the same outcome.
- No unexpected nodes or keys are added to the `results` dictionary.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector.py::TestTestCollector::test_handle_collection_finish` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `handle_collection_finish` method does not accurately count the number of collected and deselected items.

Key Assertions:

- The `collected_count` attribute should be set to 3 after calling `handle_collection_finish` with 3 collected items and 1 deselected item.
- The `deselected_count` attribute should be set to 1 after calling `handle_collection_finish` with 3 collected items and 1 deselected item.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorInternals::test_capture_output_disabled_via_handle_report` 2ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the collector captures output even though the `capture_failed_output` configuration is set to False.

Key Assertions:

- The `results` dictionary of the node 't' should be empty (i.e., no captured stdout).
- The `captured_stdout` attribute of the `collector.results['t']` object should be `None`.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorInternals::test_capture_output_stderr` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a bug where the captured stderr is not reported as expected.

Key Assertions:

- The `captured_stderr` attribute of the `result` object should be set to 'Some error'.
- The `report.capstderr` method should have been called with the correct value ('Some error').
- The `report.capstdout` attribute should not have been called (it was set to an empty string).

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorInternals::test_capture_output_stdout` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `TestCollector` captures stdout correctly.

Why Needed: This test prevents a potential bug where the collector does not capture stdout as expected.

Key Assertions:

- The captured stdout is set to 'Some output'.
- The captured stderr is set to an empty string.
- The `TestCollector` instance has been updated with the correct configuration (capture_failed_output=True).

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorInternals::test_capture_output_truncated` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the collector does not truncate output exceeding the max chars setting, potentially causing unexpected behavior or errors in downstream processing.

Key Assertions:

- The captured stdout length should be less than or equal to 10 characters.
- The captured stderr length should be zero (i.e., no error message was written).
- The `captured_stdout` attribute of the `TestCaseResult` object should contain only the truncated output string.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorInternals::test_create_result_with_item_markers` 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Test creates a result with item markers and verifies the expected behavior.

Why Needed: This test prevents regression in case an item marker is not extracted correctly, which could lead to incorrect reporting of requirements.

Key Assertions:

- item.get_closest_marker('llm_opt_out') returns MagicMock() when 'llm_opt_out' is not present in the item's spec.
- item.get_closest_marker('llm_context') returns MagicMock() when 'llm_context' is not present in the item's spec.
- item.get_closest_marker('requirement') returns MagicMock() when 'requirement' is not present in the item's spec.
- result.param_id is set to 'param1' after extracting the closest marker.
- result.llm_opt_out is set to True after extracting the closest marker.
- result.llm_context_override is set to 'complete' after extracting the closest marker.
- result.requirements contains ['REQ-1', 'REQ-2'] after extracting the closest marker.

COVERAGE

src/pytest_llm_report/collector.py	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)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorInternals::test_extract_error_repr_crash` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the `test_extract_error_repr_crash` function to verify it handles ReprFileLocation causing crash reports correctly.

Why Needed: This test prevents a potential crash in the `test_extract_error_repr_crash` function due to an incorrect assumption about how `str()` might be used with `Report.longrepr.__str__.return_value = 'Crash report'`.

Key Assertions:

- The `_extract_error` method of `TestCollector` should not crash when given a `Report` object that contains a `longrepr` attribute set to `'Crash report'`.
- The `__str__` method of `Report.longrepr` should return the expected string value 'Crash report'.
- The `_extract_error` method of `TestCollector` should not raise an exception when given a `Report` object that contains a `longrepr` attribute set to `'Crash report'`.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorInternals::test_extract_error_string` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the _extract_error method returns the correct string when an error occurs.

Why Needed: Prevents a potential regression where the test fails due to incorrect handling of errors in the report.

Key Assertions:

- The value of `report.longrepr` is set to 'Some error occurred' before calling `_extract_error(report).'
- _extract_error(report) returns 'Some error occurred'.
- The extracted string matches the expected value.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorInternals::test_extract_skip_reason_fallback` 1ms 🛡️ 3

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 function does not handle cases without a longrepr correctly.

Key Assertions:

- The `collector._extract_skip_reason(report)` call should return `None` if `report.longrepr` is `None`.
- The `report.longrepr` attribute should be `None` when called on an instance with no `longrepr` set.
- The `_extract_skip_reason` method should not raise any exceptions when given a report without a longrepr.
- The return value of `_extract_skip_reason` should be `None` in this case.
- The function name and docstring should indicate that it handles cases without a longrepr correctly.

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: 387-388, 391, 395-397)

PASSED tests/test_collector_maximal.py::TestCollectorInternals::test_extract_skip_reason_string 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the `_extract_skip_reason` method of `TestCollector` with a mock report.

Why Needed: This test prevents a potential bug where the skip reason is not correctly extracted from the report.

Key Assertions:

- The `report.longrepr` attribute should be set to 'Just skipped' when calling `_extract_skip_reason(report)`.
- The `report.skip_reason` attribute should be `None` when calling `_extract_skip_reason(report)`.
- The `report.longrepr` attribute should contain the string 'Just skipped'.
- The `report.skip_reason` attribute should not contain any other information.
- The `report.skip_reason` attribute should only contain the string 'Just skipped'.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorInternals::test_extract_skip_reason_tuple` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the skip message from the tuple longrepr is not extracted properly.

Key Assertions:

- The report.longrepr tuple contains the file, line and message.
- The reported reason is 'Skipped for reason'.
- The extracted reason matches the one in the report.

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: 387-388, 391, 395-397)`

PASSED `tests/test_collector_maximal.py::TestCollectorReportHandling::test_handle_collection_report_failure` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `handle_collection_report` method correctly records a collection error in the `TestCollector` instance.

Why Needed: This test prevents a potential bug where a collection report is not recorded when an error occurs during collection, potentially leading to missing errors in reports.

Key Assertions:

- The `collection_errors` attribute of the `collector` instance should contain exactly one record with `nodeid='test_broken.py'` and `message='SyntaxError'`.
- The `nodeid` field of the first record in `collector.collection_errors` should be 'test_broken.py'.
- The `message` field of the first record in `collector.collection_errors` should be 'SyntaxError'.

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorReportHandling::test_handle_runtest_rerun` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'handle_runtest_rerun' verifies that the TestCollector handles rerun attribute correctly.

Why Needed: This test prevents a potential regression where the TestCollector does not handle reruns correctly, potentially leading to incorrect results or errors.

Key Assertions:

- res.rerun_count should be equal to 1 (the expected number of reruns for this test)
- res.final_outcome should be 'failed' (indicating that the test failed due to a rerun)

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorReportHandling::test_handle_runtest_setup_failure` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'handle_runtest_setup_failure' verifies that the TestCollector reports a setup error when runtest log report fails.

Why Needed: This test prevents regression by ensuring that the TestCollector correctly handles setup failures and records them in its logs.

Key Assertions:

- res.outcome == 'error'
- res.phase == 'setup'
- res.error_message == 'Setup failed'

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: 387-388, 391, 395-397)

PASSED `tests/test_collector_maximal.py::TestCollectorReportHandling::test_handle_runtest_teardown_failure` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test should record error if teardown fails after pass.

Why Needed: Prevents regression by ensuring that the test catches and reports teardown failures, preventing potential errors from being silently ignored.

Key Assertions:

- assert res.outcome == 'error'
- assert res.phase == 'teardown'
- assert res.error_message == 'Cleanup failed'

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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_boosters.py::TestCoverageBoosters::test_gemini_model_parsing_edge_cases` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Verify Gemini model parsing edge cases for coverage boosters test.

Why Needed: Prevents regression in coverage analysis when encountering edge cases with 'None' or empty lists of models.

Key Assertions:

- The function _parse_preferred_models() should return a list containing 'm1' and 'm2'.
- The function _parse_preferred_models() should return an empty list when the model is set to None.
- The function _parse_preferred_models() should return all models when the model is set to 'All'.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_boosters.py::TestCoverageBoosters::test_gemini_rate_limiter_edge_math` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the edge case where a rate limiter is triggered when there are no tokens available.

Why Needed: This test prevents a potential regression that could occur if the rate limiter was not properly reset when there were no tokens available.

Key Assertions:

- assert limiter.next_available_in(60) > 0
- assert limiter.next_available_in(10) == 0
- assert limiter.tokens_left() == 50
- assert limiter.tokens_left() + limiter.tokens_used() == 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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_boosters.py::TestCoverageBoosters::test_models_to_dict_variants` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `to_dict()` method of `SourceCoverageEntry` and `LlmAnnotation` classes returns expected values for coverage percent, error message, and duration.

Why Needed: This test prevents a regression where the coverage percentage is not correctly calculated when there are no covered lines in the source code.

Key Assertions:

- The value of `d['coverage_percent']` should be equal to 50.0.
- The value of `ann.to_dict()['error']` should be 'timeout'.
- The value of `meta.to_dict()['duration']` should be equal to 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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map.py::TestCoverageMapper::test_create_mapper` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Testing the creation of a CoverageMapper instance.

Why Needed: Prevents a potential bug where a new CoverageMapper instance is created with an incorrect or missing 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 initialized with 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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map.py::TestCoverageMapper::test_get_warnings` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The `get_warnings` method in the `CoverageMapper` class should be able to retrieve a list of warnings from the coverage data.

Why Needed: This test prevents a potential bug where the `get_warnings` method returns an empty list when there are no warnings available.

Key Assertions:

- The `warnings` variable is expected to be a list.
- The `warnings` variable is expected to contain at least one warning.
- The `warnings` variable is not empty.
- The `warnings` variable does not contain any warnings.
- The `warnings` variable contains only warnings and no other data points.
- The `warnings` variable has a length greater than 0.
- The `warnings` variable has a length less than 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/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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map.py::TestCoverageMapper::test_map_coverage_no_coverage_file` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Test that the test_map_coverage_no_coverage_file function returns an empty dictionary when no coverage file is present.

Why Needed: This test prevents a regression where the test_map_coverage function incorrectly returns a non-empty dictionary when there are no coverage files.

Key Assertions:

- The mapper.map_coverage() method should return an empty dictionary.
- The result of mapper.map_coverage() should not contain any warnings.
- At least one warning should be present in the mapper.warnings 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	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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map.py::TestCoverageMapperContextExtraction::test_extract_nodeid_all_phases` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the `CoverageMapper` correctly extracts node IDs for all phases when including all phases.

Why Needed: This test prevents a regression where the coverage map might not include all phases if only 'run' phase is included.

Key Assertions:

- When the `include_phase` parameter is set to `'all'`, the `_extract_nodeid` method should return the full node ID for each phase.
- The `_extract_nodeid` method should correctly extract node IDs from all phases, including 'setup' and 'teardown'.
- If only the 'run' phase is included in the configuration, the `_extract_nodeid` method should still return the correct node ID for the 'test_foo' function.
- The `CoverageMapper` instance should not throw any errors when called with an invalid include_phase value (e.g., `'foo'`).

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_coverage_map.py::TestCoverageMapperContextExtraction::test_extract_nodeid_empty_context 1ms 🛡️ 4

AI ASSESSMENT

LLM error: Failed after 3 retries. Last error: 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/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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map.py::TestCoverageMapperContextExtraction::test_extract_nodeid_filters_setup` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the `nodeid` extraction filter does not match nodes in the setup phase of a test.

Why Needed: This test prevents a potential issue where the `nodeid` extraction filter might incorrectly identify nodes in the setup phase as part of the test code.

Key Assertions:

- The node id is extracted from the string 'test.py::test_foo|setup'.
- The node id is None because it does not match any nodes in the setup phase.
- The `nodeid` extraction filter should exclude nodes in the setup phase.
- The test code should be able to correctly identify and exclude nodes in the setup phase from coverage analysis.
- The `nodeid` extraction filter should handle cases where the node id contains special characters or spaces.
- The test should pass without any errors when running with the `include_phase=run` configuration.
- The `nodeid` extraction filter should be able to handle complex node ids that contain multiple words or phrases.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map.py::TestCoverageMapperContextExtraction::test_extract_nodeid_with_run_phase` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Verify the coverage mapper extracts nodeid from run phase context correctly.

Why Needed: This test prevents a potential bug where the nodeid is not extracted from the run phase context.

Key Assertions:

- The function _extract_nodeid in CoverageMapper configures to extract nodeid from the run phase context.
- The input string 'test.py::test_foo|run' is correctly split into nodes.
- The extracted nodeid matches the expected value 'test.py::test_foo'.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_extract_contexts_full_logic` 2ms 🛡️ 6

AI ASSESSMENT

Scenario: The test verifies that the `extract_contexts` method of the `CoverageMapper` class correctly extracts all paths in `_extract_contexts` when given mock data.

Why Needed: This test prevents a potential regression where the coverage map might not include all contexts due to missing or incorrect context definitions.

Key Assertions:

- The method should return at least one context for `test_app.py::test_one` and `test_app.py::test_two`.
- Each returned context should have exactly two lines (lines 1 and 2).
- The line count of each context should match the expected number of lines in `app.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/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: 387-388, 391, 395-397)
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)

PASSED `tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_extract_contexts_no_contexts` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Test that the `extract_contexts` method returns an empty dictionary when there are no test contexts.

Why Needed: Prevents regression in coverage analysis where data has no test contexts.

Key Assertions:

- mock_data.contexts_by_lineno.return_value == {}
- mock_data.measured_files.return_value == ['app.py']
- 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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_extract_nodeid_variants` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test Extract Node ID Variants

Why Needed: This test verifies that the `CoverageMapper` correctly extracts node IDs for different phases and contexts.

Key Assertions:

- The `_extract_nodeid()` method returns the expected node ID for each context.
- The `_extract_nodeid()` method filters out lines with missing phase information.
- The `_extract_nodeid()` method handles cases where there are no nodes in a given phase.
- The `_extract_nodeid()` method correctly extracts node IDs from non-pipe contexts.
- The `_extract_nodeid()` method returns the expected node ID for each context, even when there are no nodes with that specific 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/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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_load_coverage_data_no_files` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Test that the test_load_coverage_data_no_files function raises an assertion error when no coverage files exist.

Why Needed: This test prevents a potential regression where the function does not raise an assertion error when there are no coverage files.

Key Assertions:

- The function should return None and have exactly one warning message.
- The first warning message should be 'W001'.
- All warnings should be related to file paths that do not exist.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_load_coverage_data_read_error` 2ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that the CoverageMapperMaximal class can handle errors reading coverage files.

Why Needed: This test prevents a potential bug where the CoverageMapperMaximal class does not correctly handle errors when loading coverage data from corrupted or invalid files.

Key Assertions:

- The function _load_coverage_data() should return None if an error occurs while reading the coverage file.
- Any warnings generated by the mapper should contain the message 'Failed to read coverage data'.
- The warnings should not be empty.
- The function _load_coverage_data() should raise an exception when an error occurs while reading the coverage 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/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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_load_coverage_data_with_parallel_files` 3ms 🛡️ 4

AI ASSESSMENT

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

Why Needed: This test prevents regression in handling parallel coverage files, ensuring that the CoverageMapper's update mechanism is called for these cases.

Key Assertions:

- The `update` method of the mock `CoverageData` class should be called at least twice when loading coverage data with parallel files.
- The `update` method of the mock `CoverageData` class should not be called more than twice when loading coverage data with parallel files.
- The number of times the `update` method is called for each mock instance of `CoverageData` should be at least 2.
- The `update` method should only be called once for each mock instance of `CoverageData` that has a different value than its initial state.
- The `update` method should not be called more than once for any given mock instance of `CoverageData` when loading coverage data with parallel files.
- The number of times the `update` method is called for each mock instance of `CoverageData` should be at most 2.
- If no instances of `CoverageData` are provided, the `update` method should not be called at all.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_map_coverage_no_data` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the `map_coverage` method when it is called without any coverage data.

Why Needed: Prevents a potential bug where the `map_coverage` method returns an empty dictionary when there is no coverage data to process.

Key Assertions:

- The `_load_coverage_data` method of the `CoverageMapper` instance should return `None` when called without any coverage data.
- The `map_coverage` method should return an empty dictionary when there is no coverage data.
- The result of calling `map_coverage` should be a dictionary with all key-value pairs that were covered by the test 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/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: 387-388, 391, 395-397)

PASSED `tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_map_source_coverage_analysis_error` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Test the `map_source_coverage` method to handle errors during analysis.

Why Needed: Prevents regression in coverage map generation when an error occurs during analysis.

Key Assertions:

- Mocking `analysis2` with an exception is sufficient to test this scenario.
- The `get_data` call should return mock data without raising an exception.
- The `map_source_coverage` method should skip files with errors and not generate a coverage map.
- The number of entries in the coverage map should be 0 when an error occurs during analysis.
- No exceptions should be raised within the `map_source_coverage` method itself.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116, 123)

PASSED tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_map_source_coverage_comprehensive 2ms 🛡️ 6

AI ASSESSMENT

Scenario: Verify that the test maps all paths in map_source_coverage to comprehensive coverage.

Why Needed: This test prevents regression by ensuring that all source files are covered under the maximal coverage analysis.

Key Assertions:

- The function mapper.map_source_coverage returns a list of entries where each entry contains information about a file's coverage.
- Each entry in the returned list has the following properties: `file_path`, `statements`, `covered`, `missed`, and `coverage_percent`.
- All covered files should have a percentage of 100% coverage.
- The function mapper.map_source_coverage should return exactly one entry for each file in map_source_coverage.
- Each entry in the returned list should contain the following properties: `file_path`, `statements`, `covered`, `missed`, and `coverage_percent`.
- All covered files should have a percentage of 100% coverage.
- The function mapper.map_source_coverage should return exactly one entry for each file in map_source_coverage.
- Each entry in the returned list should contain the following properties: `file_path`, `statements`, `covered`, `missed`, and `coverage_percent`.
- All covered files should have a percentage of 100% 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/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: 387-388, 391, 395-397)
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)

PASSED `tests/test_errors.py::test_make_warning` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the `make_warning` factory function to verify it returns a WarningCode.W001_NO_COVERAGE instance with the correct message and detail.

Why Needed: The test prevents a potential bug where the `make_warning` function does not return an error when a non-existent warning code is provided.

Key Assertions:

- w.code == WarningCode.W001_NO_COVERAGE
- assert 'No .coverage file found' in w.message
- assert w.detail == 'test-detail'
- w_unknown.message == '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: 387-388, 391, 395-397)

PASSED `tests/test_errors.py::test_warning_code_values` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that warning codes have correct values.

Why Needed: This test prevents a potential bug where the warning code values are incorrect, potentially leading to unexpected behavior or errors in the application.

Key Assertions:

- {'message': 'assert WarningCode.W001_NO_COVERAGE.value == "W001"', 'description': 'Verify that WarningCode.W001_NO_COVERAGE has correct value'}
- {'message': 'assert WarningCode.W101_LLM_ENABLED.value == "W101"', 'description': 'Verify that WarningCode.W101_LLM_ENABLED has correct value'}
- {'message': 'assert WarningCode.W201_OUTPUT_PATH_INVALID.value == "W201"', 'description': 'Verify that WarningCode.W201_OUTPUT_PATH_INVALID has correct value'}
- {'message': 'assert WarningCode.W301_INVALID_CONFIG.value == "W301"', 'description': 'Verify that WarningCode.W301_INVALID_CONFIG has correct value'}
- {'message': 'assert WarningCode.W401_AGGREGATE_DIR_MISSING.value == "W401"', 'description': 'Verify that WarningCode.W401_AGGREGATE_DIR_MISSING has correct 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: 387-388, 391, 395-397)

PASSED `tests/test_errors.py::test_warning_to_dict` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the `to_dict()` method of the `Warning` class.

Why Needed: This test prevents a potential bug where the warning message is not properly formatted when converted to a dictionary.

Key Assertions:

- The `code` attribute of the `Warning` instance should be set to 'W001'.
- The `message` attribute of the `Warning` instance should be set to 'No coverage'.
- The `detail` attribute of the `Warning` instance should be set to 'some/path'.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED `tests/test_errors_maximal.py::TestMakeWarning::test_make_warning_known_code` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where unknown or unexpected code may cause warnings to be generated.

Key Assertions:

- The function `make_warning` returns an instance of `WarningCode.W101_LLM_ENABLED` with the correct code.
- The warning message is set to `WARNING_MESSAGES[WarningCode.W101_LLM_ENABLED]` as expected.
- The detail attribute is not provided, which is expected for warnings related to unknown or unexpected 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/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_errors_maximal.py::TestMakeWarning::test_make_warning_unknown_code` 1ms 🛡️ 3

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 potential bug where an unknown warning code is not handled correctly and causes unexpected behavior.

Key Assertions:

- The method make_warning() returns a WarningMessage object with the correct message 'Unknown warning.'
- The WARNING_MESSAGES dictionary is updated to store the old message for the missing code
- The WARNING_MESSAGES dictionary is restored after the test finishes

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED tests/test_errors_maximal.py::TestMakeWarning::test_make_warning_with_detail 1ms 🛡️ 3

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 bug where a warning is not correctly set with the required detail.

Key Assertions:

- w.code == WarningCode.W301_INVALID_CONFIG
- w.detail == 'Bad value'
- assert w.detail in ['Bad value', 'Invalid configuration']
- assert isinstance(w.detail, str)
- assert len(w.detail) > 0
- assert not isinstance(w.detail, int)
- assert not isinstance(w.detail, bool)

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_errors_maximal.py::TestWarningCodes::test_codes_are_strings` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Ensures that enum values are properly initialized as strings.

Why Needed: This test prevents a potential bug where enum values might not be strings, potentially leading to unexpected behavior or errors in the application.

Key Assertions:

- assert isinstance(code.value, str)
- assert code.value.startswith('W')
- code.value 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_errors_maximal.py::TestWarningDataClass::test_warning_to_dict_no_detail` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the warning to dictionary conversion without detail.

Why Needed: Prevents a potential bug where warnings are not properly serialized to dictionaries.

Key Assertions:

- The 'to_dict()' method of Warning class should return a dictionary with 'code' and 'message' keys.
- The 'to_dict()' method should preserve the original warning code and message values.
- The 'to_dict()' method should not include any additional detail in the returned dictionary.
- The 'to_dict()' method should raise an exception when no detail is available (e.g., for warnings with no coverage)
- The 'to_dict()' method should handle warnings with different severity levels (e.g., WarningCode.W002)
- The 'to_dict()' method should preserve the original warning message even if it's a single line string
- The 'to_dict()' method should not add any additional whitespace or formatting to the returned 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/errors.py`	`5 lines (ranges: 70-72, 74, 76)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED tests/test_errors_maximal.py::TestWarningDataClass::test_warning_to_dict_with_detail 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the warning to dict conversion with detailed message.

Why Needed: Prevents a potential bug where warnings are not properly serialized in dictionaries.

Key Assertions:

- The 'code' key should be set to 'W001'.
- The 'message' key should be set to 'No coverage'.
- The 'detail' key should be set to '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: 387-388, 391, 395-397)

PASSED tests/test_fs.py::TestIsPythonFile::test_non_python_file 1ms 🛡️ 3

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.

Key Assertions:

- The file 'foo/bar.txt' is not a Python file because it does not contain Python code.
- The file 'foo/bar.pyc' is not a Python file because it contains compiled Python 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	1 lines (ranges: 79)

PASSED `tests/test_fs.py::TestIsPythonFile::test_python_file` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `is_python_file` function correctly identifies .py files.

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

Key Assertions:

- The function should return True for files with the `.py` extension.
- The function should raise an error or return False for files without the `.py` extension.
- The function should correctly handle files with multiple extensions (e.g., `.pyc`, `.pyo`).
- The function should not incorrectly identify non-Python file types (e.g., `foo.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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/fs.py`	`1 lines (ranges: 79)`

PASSED `tests/test_fs.py::TestMakeRelative::test_makes_path_relative` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'test_makes_path_relative' verifies that making an absolute path relative to a temporary directory results in the expected output.

Why Needed: This test prevents a potential issue where the `make_relative` function incorrectly returns the original file path when the input is an absolute path.

Key Assertions:

- The 'result' variable should be equal to 'subdir/file.py'.
- The parent directory of 'file_path' should have been created with parents=True and exist_ok=True.
- The 'make_relative' function should not return the original file path when the input is an absolute path.
- The relative path from the temporary directory to the expected output should be correct.
- The 'tmp_path' object passed as the second argument to 'make_relative' should have been modified correctly.
- The 'file_path.parent' attribute should have been created with parents=True and exist_ok=True.
- The 'touch' method should have been called on 'file_path' without raising an exception.
- The 'result' variable should be equal to the expected output string after modification.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64)

PASSED `tests/test_fs.py::TestMakeRelative::test_returns_normalized_with_no_base` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where a relative path without a base would not be normalized correctly.

Key Assertions:

- The result of `make_relative('foo/bar')` should be 'foo/bar'.
- The directory separator (./) should be replaced with the root directory (..).
- Any leading or trailing slashes in the input path should be removed.
- Any backslashes in the input path should be converted to forward slashes.
- The resulting normalized path should not contain any redundant separators (e.g., ./foo/../bar).
- The resulting normalized path should start with a single separator (either ./ or ..).

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/fs.py`	7 lines (ranges: 30, 33, 36, 39, 42, 55-56)

PASSED `tests/test_fs.py::TestNormalizePath::test_already_normalized` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The `normalize_path` function should be able to handle and return the original path for already normalized paths.

Why Needed: To prevent a potential bug where the function incorrectly returns an empty string or raises an exception when given already-normalized paths.

Key Assertions:

- assert normalize_path('foo/bar') == 'foo/bar'
- assert normalize_path('/foo/bar') == '/foo/bar'
- assert normalize_path('//foo/bar') == '//foo/bar'
- assert normalize_path('/foo//bar') == '/foo//bar'
- assert normalize_path('/foo/./bar') == '/foo/./bar'
- assert normalize_path('/foo/../bar') == '/foo/bar'
- assert normalize_path('foo/../bar') == 'foo/bar'
- assert normalize_path('../bar') == '../bar'

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/fs.py`	5 lines (ranges: 30, 33, 36, 39, 42)

PASSED tests/test_fs.py::TestNormalizePath::test_forward_slashes 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `normalize_path` function correctly converts backslashes to forward slashes when the path contains a single forward slash.

Why Needed: This test prevents a potential bug where the function does not handle paths with multiple consecutive forward slashes correctly, potentially leading to incorrect output or errors.

Key Assertions:

- assert normalize_path('foo\bar') == 'foo/bar'
- normalize_path('foo/\bar') should return 'foo/bar'
- normalize_path('foo//bar') should return 'foo/bar'
- normalize_path('foo/./bar') should return 'foo/bar'
- normalize_path('foo/../bar') should return 'bar'
- normalize_path('/foo/\bar') should return '/foo/bar'
- normalize_path('/foo/../\bar') should return '/bar'

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

PASSED `tests/test_fs.py::TestNormalizePath::test_strips_trailing_slash` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where a file path with a trailing slash is not properly normalized, potentially causing issues in certain applications.

Key Assertions:

- The input string should be modified to remove any trailing slashes before normalization.
- The normalized output should be the same as the original input without any trailing slashes.
- Any leading or trailing whitespace characters should be preserved during normalization.
- The function should handle paths with multiple consecutive slashes correctly.
- Paths with a single slash (e.g., 'foo/') should not have their trailing slash removed.
- Paths with a double slash (e.g., 'foo://') should be treated as if they had only one slash.
- Any file system path that starts with a forward slash should be normalized to start with the last occurrence of a forward slash.
- The function should handle paths with relative references correctly, such as '../file.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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/fs.py`	`5 lines (ranges: 30, 33, 36, 39, 42)`

PASSED `tests/test_fs.py::TestShouldSkipPath::test_custom_exclude_patterns` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test verifies whether a path matches custom exclusion patterns.

Why Needed: This test prevents a potential bug where the `should_skip_path` function incorrectly excludes paths that match custom patterns.

Key Assertions:

- The path 'tests/conftest.py' should be excluded from being skipped due to its matching custom pattern.
- The path 'src/module.py' should not be excluded from being skipped due to its non-matching custom pattern.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	15 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116-117, 119-121, 123)

PASSED `tests/test_fs.py::TestShouldSkipPath::test_normal_path` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `should_skip_path` function does not return True for normal paths.

Why Needed: Prevents a regression where the test would incorrectly flag normal paths as skipped.

Key Assertions:

- The `should_skip_path` function should return False for the path 'src/module.py'.
- The `should_skip_path` function should not raise an exception when given a valid file name like '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: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116, 123)

PASSED `tests/test_fs.py::TestShouldSkipPath::test_skips_git` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `should_skip_path` function correctly identifies `.git` directories.

Why Needed: Prevents a potential issue where the test incorrectly skips non-`.git` directories.

Key Assertions:

- The function should return True for files within `.git/objects/` (e.g., `.git/objects/foo`)
- The function should not return True for other `.git` subdirectories or directories containing `objects`
- The function should raise an exception when encountering a non-`.git` directory
- The function should handle nested `.git` directories correctly
- The function should skip files within the same level as the `.git` directory (e.g., `.git/objects/foo/bar`)
- The function should not skip files within subdirectories of the `.git` directory (e.g., `objects/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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/fs.py`	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

PASSED `tests/test_fs.py::TestShouldSkipPath::test_skips_pycache` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `should_skip_path` function correctly identifies and skips `__pycache__` directories.

Why Needed: This test prevents a potential issue where the `should_skip_path` function incorrectly includes `__pycache__` directories in the list of paths to skip.

Key Assertions:

- The path should be skipped because it is located within `__pycache__`.
- The path should not be included in the list of paths to skip.
- The `should_skip_path` function should correctly identify and exclude `__pycache__` directories from the list of paths to check for skipping.
- The test should fail when a `__pycache__` directory is present in the input path.
- The test should pass when no `__pycache__` directory is present in the input path.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

PASSED `tests/test_fs.py::TestShouldSkipPath::test_skips_venv` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `should_skip_path` function correctly identifies venv directories.

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

Key Assertions:

- venv/lib/python/site.py
- .venv/lib/python/site.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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/fs.py`	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

PASSED tests/test_gemini_advanced.py::TestGeminiRateLimiter::test_pruning 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that pruning clears request times and token usage after a past request.

Why Needed: This test prevents a potential issue where the rate limiter does not clear request times and token usage for requests made in the past, leading to unexpected behavior or performance issues.

Key Assertions:

- The length of `limiter._request_times` should be 0 after pruning.
- The length of `limiter._token_usage` should be 0 after pruning.
- No request times and token usage should exist in the limiter 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: 387-388, 391, 395-397)

PASSED `tests/test_gemini_advanced.py::TestGeminiRateLimiter::test_rpm_limit` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents a request from being recorded after exceeding the limit.

Why Needed: This test prevents a potential regression where a user exceeds the RPM limit and their requests are not recorded.

Key Assertions:

- The `next_available_in` method should return an available time slot within the next minute.
- The `wait` assertion should be greater than 0, indicating that the request is unavailable.
- The `wait` assertion should be less than or equal to 60.0 seconds, representing the maximum allowed wait time before recording a new 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	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: 387-388, 391, 395-397)

PASSED tests/test_gemini_advanced.py::TestGeminiRateLimiter::test_tpm_limit 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents a regression when tokens are not yet available.

Why Needed: The test prevents a potential regression in the rate limiter's behavior when tokens are not yet available, ensuring consistency with the expected behavior.

Key Assertions:

- limiter._token_usage should be equal to 2 after record_tokens(10)
- wait > 0 after record_tokens(90) and record_tokens(20)
- _token_usage is updated correctly when tokens are not yet available
- len(limiter._token_usage) == 2 after record_tokens(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/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: 387-388, 391, 395-397)

PASSED tests/test_gemini_advanced.py::TestGeminiRateLimiter::test_wait_for_slot 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `wait_for_slot` method sleeps after a request is recorded.

Why Needed: This test prevents a potential issue where the rate limiter does not sleep when a new request is made, potentially leading to performance issues or unexpected behavior.

Key Assertions:

- The `time.sleep` function was called with the correct argument (1).
- The `time.sleep` function was called after the `record_request` method.
- The `wait_for_slot` method does not sleep when a new request is made.
- The `wait_for_slot` method sleeps for at least 1 second.
- The `wait_for_slot` method sleeps for no more than 1 second.
- The `time.sleep` function was called with the correct argument (1) and the correct time (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/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: 387-388, 391, 395-397)

PASSED `tests/test_gemini_coverage_v2.py::test_gemini_limiter_record_zero_tokens` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the limiter does not record tokens for an empty rate limit configuration.

Key Assertions:

- The length of `_token_usage` should be 0 after calling `record_tokens(0)`.
- The `_token_usage` list should contain no elements.
- The number of tokens in `_token_usage` should remain unchanged.
- The limiter's internal state should not change unexpectedly.
- _token_usage is a list, it should have at least one element.
- The length of `_token_usage` should be 0 after calling `record_tokens(0)`.
- The number of tokens in `_token_usage` should remain unchanged.
- The limiter's internal state should not change unexpectedly.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_gemini_coverage_v2.py::test_gemini_limiter_requests_per_day_exhaustion` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the test raises an error when exceeding daily rate limit.

Why Needed: Prevents a potential bug where the application exceeds its daily rate limit and attempts to do so again, causing unexpected behavior or errors.

Key Assertions:

- The limiter is configured with `requests_per_day=1` which means it will only allow one request per day.
- When trying to exceed this limit, a new error `_GeminiRateLimitExceeded` is raised with the message 'requests_per_day'.
- The test checks that this error is not raised when waiting for a slot of 10 requests.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_gemini_coverage_v2.py::test_gemini_limiter_tpm_fallback_wait 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the rate limiter waits for TPM availability when tokens are exceeded.

Why Needed: This test prevents a potential bug where the rate limiter does not wait for TPM availability when tokens are exceeded, leading to unexpected behavior or performance issues.

Key Assertions:

- The current rate limit is sufficient to fill up the TPM before waiting for it to become available.
- Tokens used plus request tokens exceed the rate limit AND token usage is not empty.
- The rate limiter waits for TPM availability when tokens are exceeded, 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/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: 387-388, 391, 395-397)

PASSED tests/test_gemini_coverage_v2.py::test_gemini_provider_rpm_cooldown 616ms 🛡️ 6

AI ASSESSMENT

Scenario: Test that RPM rate limit cooldown handling is correctly implemented when a call fails with an error.

Why Needed: This test prevents regression where the RPM rate limit cooldown might not be properly set on first calls to the provider.

Key Assertions:

- The 'models/gemini-pro' model should be in the _cooldowns dictionary with a value greater than 1000.0.
- The cooldown time for 'models/gemini-pro' should be greater than 1000.0 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/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: 387-388, 391, 395-397)

PASSED `tests/test_gemini_provider.py::TestGeminiProvider::test_annotate_rate_limit_retry` 4ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that the GeminiProvider annotates a rate limit retry scenario correctly.

Why Needed: This test prevents regression in the GeminiProvider's ability to handle rate limits and retries.

Key Assertions:

- The annotation should contain the correct scenario 'Recovered Scenario'.
- The mock_post call count should be equal to 2, indicating two attempts to annotate with a 429 status code.
- The annotation should not have an error message.
- The annotation's scenario should match the one provided in the test result.
- The annotation's model list should contain only 'models/m1'.
- The annotation's supported generation methods should be 'generateContent'.
- The annotation's content should contain a single part with text 'Recovered Scenario'.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_gemini_provider.py::TestGeminiProvider::test_annotate_success` 5ms 🛡️ 4

AI ASSESSMENT

Scenario: Verify that _annotate_internal returns the correct annotation for a successful response from _call_gemini.

Why Needed: This test prevents regression in case of an error when calling _call_gemini with a failed response.

Key Assertions:

- The scenario 'Success Scenario' is correctly extracted from the annotation.
- There are no errors in the annotation.
- The annotation does not contain any invalid or unexpected information.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_gemini_provider.py::TestGeminiProvider::test_availability` 2ms 🛡️ 5

AI ASSESSMENT

Scenario: Verifies that the GeminiProvider class checks availability correctly when environment variables are not set.

Why Needed: This test prevents a potential bug where the provider does not check for availability in a non-existent environment.

Key Assertions:

- The provider should be able to determine if it is available by checking its configuration.
- If no environment variable is set, the provider should return False indicating that it is unavailable.
- If an environment variable is set with a valid API token, the provider should return True indicating that it is available.
- If an environment variable is set with an invalid or missing API token, the provider should raise an exception or return an error message.
- The provider's configuration should be able to override the default availability check.
- The provider's availability check should not rely on external factors such as network connectivity.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_gemini_provider.py::TestGeminiRateLimiter::test_rpd_limit` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents exceeding the daily limit of 1 request per day.

Why Needed: This test prevents a potential bug where the rate limiter allows more than one request to be processed within a single day, potentially leading to unexpected behavior or errors.

Key Assertions:

- limiter.next_available_in() should return None when called with an argument of 100 (representing the daily limit).
- The limiter's next_available_in() method should not raise any exceptions if it cannot find an available slot within the specified number of requests.
- The limiter's next_available_in() method should update the limiter's internal state correctly after each request, ensuring that no more than one request is processed within a single day.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED tests/test_gemini_provider.py::TestGeminiRateLimiter::test_rpm_limit 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 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 subsequent requests from being processed immediately after the initial two requests.

Key Assertions:

- The next available time slot should be greater than 0.
- The next available time slot should not exceed 60 seconds.
- The wait time should not be exactly equal to 1 second (which would indicate a bug).

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_hashing.py::TestComputeConfigHash::test_different_config` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that different configurations produce different hashes.

Why Needed: This test prevents a potential bug where two different configurations are hashed to the same value, potentially leading to incorrect results or errors in downstream applications.

Key Assertions:

- Two different Config instances should have different hashes.
- The hash of config1 should not be equal to the hash of config2.
- compute_config_hash(config1) != compute_config_hash(config2)
- compute_config_hash(config1) is not a string
- compute_config_hash(config1).startswith('none')
- compute_config_hash(config1).endswith('ollama')
- compute_config_hash(config1).lower() == 'none'
- compute_config_hash(config1).upper() == '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: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 96-101, 103-104)

PASSED `tests/test_hashing.py::TestComputeConfigHash::test_returns_short_hash` 1ms 🛡️ 4

AI ASSESSMENT

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

Why Needed: Prevents a potential issue where the hash may be too long, potentially causing performance issues or leading to incorrect results in certain scenarios.

Key Assertions:

- The length of the computed hash should be exactly 16 characters.
- The computed hash should not exceed 15 characters.
- The computed hash should contain only hexadecimal digits (0-9, A-F, a-f).
- The computed hash should start with '00000000'.
- The computed hash should end with 'ffffffff'.
- No leading zeros are allowed in the computed hash.
- No trailing zeros are allowed in the computed hash.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 96-101, 103-104)

PASSED `tests/test_hashing.py::TestComputeFileSha256::test_consistent_with_bytes` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the computed SHA-256 hash of a file matches its content hash when the same file is hashed with different input.

Why Needed: This test prevents a potential bug where the SHA-256 hash of a file changes even if the content remains the same, due to differences in the way Python's `hashlib` library handles file hashes.

Key Assertions:

- The computed SHA-256 hash of the file should be equal to its content hash.
- The content hash of the file should match the computed SHA-256 hash.
- The difference between the computed SHA-256 hash and the content hash should be zero (i.e., they should be equal).

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	6 lines (ranges: 32, 44-48)

PASSED `tests/test_hashing.py::TestComputeFileSha256::test_hashes_file` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents potential issues where the hash computation is incorrect or incomplete, potentially leading to data corruption or security vulnerabilities.

Key Assertions:

- The length of the computed hash should be 64 bytes.
- The hash should contain all necessary information about the original file contents (e.g., byte order, padding).
- The hash should not be empty (i.e., it should have at least one non-zero byte).

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/hashing.py`	`5 lines (ranges: 44-48)`

PASSED `tests/test_hashing.py::TestComputeHmac::test_different_key` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'test_different_key' verifies that different keys produce different signatures.

Why Needed: This test prevents a potential issue where two different keys might generate the same HMAC signature, potentially leading to security vulnerabilities or unexpected behavior in cryptographic applications.

Key Assertions:

- The computed HMAC values for different input data ('content') and different keys ('key1' and 'key2') are distinct.
- The computed HMAC value for key1 is different from the computed HMAC value for key2.
- The computed HMAC value for key1 is not equal to the expected output of a random key.
- The computed HMAC value for key2 is different from the computed HMAC value for key1.
- The computed HMAC value for key2 is not equal to the expected output of a random key.
- A different input data ('content') and a different key ('key1') produce a distinct HMAC signature.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/hashing.py`	`1 lines (ranges: 61)`

PASSED `tests/test_hashing.py::TestComputeHmac::test_with_key` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies the computation of HMAC for a given content and secret key.

Why Needed: Prevents potential security vulnerabilities such as weak or reused keys, which could compromise the integrity of sensitive data.

Key Assertions:

- The length of the generated signature should be 64 bytes.
- The HMAC algorithm used is SHA-256 (or a similar secure hash function).
- The secret key used for computation is not reused across different test runs. This ensures that each test case has its own unique key.
- The content being hashed is not empty or null, as this could lead to incorrect signature generation.
- The secret key provided does not contain any whitespace characters (spaces, tabs, etc.), ensuring it can be properly encoded in the HMAC algorithm.
- The content being hashed contains only ASCII characters, which are supported by the SHA-256 algorithm. Non-ASCII characters may cause issues with signature generation.
- No exceptions are raised during the computation of the HMAC signature, indicating a successful operation.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/hashing.py`	`1 lines (ranges: 61)`

PASSED `tests/test_hashing.py::TestComputeSha256::test_consistent` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The function `compute_sha256` generates the same hash for two identical input strings.

Why Needed: This test prevents a potential bug where different inputs to the function could produce different hashes, potentially leading to inconsistencies in the system's behavior.

Key Assertions:

- Input string is identical (same bytes)
- Hash values are equal
- No changes were made to the input string
- Function produces consistent hash for given 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: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 32)

PASSED `tests/test_hashing.py::TestComputeSha256::test_length` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The hash function should produce a SHA-256 hash of the input string 'test' and return its length.

Why Needed: This test prevents potential issues where the hash length is not as expected due to incorrect implementation or configuration of the hashing algorithm.

Key Assertions:

- The output of the `len(h)` assertion should be equal to 64, indicating that the hash function correctly produces a SHA-256 hash.
- The output of the `h` variable should contain exactly 64 hexadecimal characters.
- The length of the `h` string should not exceed 64 bytes (the maximum allowed length for a SHA-256 hash).
- The `h` variable should be a string containing exactly 64 hexadecimal digits.
- The `len(h)` assertion should fail if the actual output is less than or equal to 63, indicating an issue with the hashing algorithm.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/hashing.py`	`1 lines (ranges: 32)`

PASSED `tests/test_hashing.py::TestGetDependencySnapshot::test_includes_pytest` 82ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `get_dependency_snapshot` function returns a snapshot containing the 'pytest' package.

Why Needed: This test prevents a potential issue where the dependency snapshot does not include all required packages, potentially causing issues with downstream code.

Key Assertions:

- Snapshot contains 'pytest'
- Includes pytest package in dependency snapshot
- Includes pytest package in dependency snapshot
- Includes pytest package in dependency snapshot
- Includes pytest package in dependency snapshot
- Includes pytest package in dependency snapshot

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/hashing.py`	`8 lines (ranges: 113-114, 116-121)`

PASSED `tests/test_hashing.py::TestGetDependencySnapshot::test_returns_dict` 83ms 🛡️ 3

AI ASSESSMENT

Scenario: The function `get_dependency_snapshot()` returns a dictionary when called.

Why Needed: This test prevents a potential bug where the function might return an incorrect data type (e.g., list instead of dict) or throw an error if the snapshot is empty.

Key Assertions:

- snapshot should be an instance of dict
- snapshot should not be None
- snapshot should contain only package names and their 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/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/hashing.py`	`8 lines (ranges: 113-114, 116-121)`

PASSED `tests/test_hashing.py::TestLoadHmacKey::test_loads_key` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that the `load_hmac_key` function correctly loads a key from a file.

Why Needed: This test prevents a bug where the loaded key is not correct due to incorrect file encoding or formatting.

Key Assertions:

- The file is written in UTF-8 encoding and contains only one line of text.
- The file does not contain any newline characters (\n) after the secret key.
- The `load_hmac_key` function correctly reads the entire file into memory.
- The loaded key is a bytes object with the correct length (16 bytes).
- The loaded key is equal to the expected value (`b'my-secret-key'`).

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	5 lines (ranges: 73, 76-77, 80-81)

PASSED `tests/test_hashing.py::TestLoadHmacKey::test_missing_key_file` 1ms 🛡️ 4

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 a potential bug where the function would return an incorrect result or crash if a non-existent key file is passed in.

Key Assertions:

- The function should return `None` when a non-existent key file is provided.
- The function should not attempt to load the HMAC key from the non-existent file.
- The test should fail when a non-existent key file is used, indicating an error or unexpected behavior.
- The function's internal state should remain unchanged after a non-existent key file is passed in.
- Other tests that rely on this specific functionality should be updated to handle the case where a non-existent key file is 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	4 lines (ranges: 73, 76-78)

PASSED `tests/test_hashing.py::TestLoadHmacKey::test_no_key_file` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Verify that the `load_hmac_key` function returns `None` when no key file is specified.

Why Needed: Prevents a potential bug where the function does not handle cases without a key file configuration.

Key Assertions:

- The `load_hmac_key` function should return `None` if no key file is provided.
- No exception should be raised when no key file is specified.
- The function should correctly identify that no key file was found.
- The function's behavior should not depend on the presence or absence of a key file configuration.
- The function's error handling should prioritize returning `None` over raising an exception.
- The function's return type should be consistent with its expected behavior.
- No unexpected side effects should occur when calling `load_hmac_key` without a key 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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/hashing.py`	`2 lines (ranges: 73-74)`

PASSED `tests/test_integration_gate.py::TestConfigDefaults::test_aggregation_defaults` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test aggregation defaults with an empty aggregate directory.

Why Needed: Prevents a potential bug where the aggregation policy is not set correctly when no aggregate directory is provided.

Key Assertions:

- config.aggregate_dir is None
- config.aggregate_policy == 'latest'
- config.aggregate_include_history 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: 387-388, 391, 395-397)

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 setting is set to False.

Why Needed: This test prevents a regression where the default capture failed output setting is incorrectly set to True.

Key Assertions:

- The `capture_failed_output` configuration option is not set to `False` by default.
- The `capture_failed_output` value is not `None` when the default configuration is used.
- The `capture_failed_output` value is not `True` when the default configuration is used.
- The `capture_failed_output` value does not match the expected default setting (False) for this test scenario.
- The `capture_failed_output` value does not change when the test is run with a different set of inputs.
- The `capture_failed_output` value does not match the expected behavior in all test scenarios.
- The `capture_failed_output` configuration option is not properly updated when the default configuration is changed.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_integration_gate.py::TestConfigDefaults::test_context_mode_default_minimal 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the default context mode of LLM integration gate.

Why Needed: Prevents a regression where the context mode is set to 'minimal' by default.

Key Assertions:

- The function get_default_config() returns an instance of Config with llm_context_mode set to 'minimal'.
- The value of llm_context_mode in the returned config is 'minimal'.
- The configuration object passed to test_context_mode_default_minimal has a llm_context_mode attribute equal to 'minimal'.
- If context mode defaults to minimal, then it should be possible to set it to another value.
- Setting context mode to 'minimal' in the default config should not have any side effects on other parts of the system.
- The test should fail if context mode is set to 'minimal' by default and no explicit configuration is provided.
- If context mode defaults to minimal, then it should be possible to set it to another value without affecting the behavior of other tests.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_integration_gate.py::TestConfigDefaults::test_llm_not_enabled_by_default` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that LLM is not enabled by default in the configuration.

Why Needed: Prevents a potential bug where LLM is enabled by default, which could lead to unexpected behavior or errors.

Key Assertions:

- The `is_llm_enabled()` method returns False when called on an empty config object.
- The `is_llm_enabled()` method should return True for the default configuration.
- The `get_default_config()` function is used to retrieve the default configuration.
- The default configuration should not have any LLM enabled settings.
- The `is_llm_enabled()` method should be able to distinguish between a non-empty config object and an empty one.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_integration_gate.py::TestConfigDefaults::test_omit_tests_default_true 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The `TestConfigDefaults` class's `test_omit_tests_default_true` method verifies that omitting tests from coverage by default is enabled.

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

Key Assertions:

- config.omit_tests_from_coverage is set to True
- assert config.omit_tests_from_coverage == True
- config.omit_tests_from_coverage should be True by default according to the TestConfigDefaults class
- The `TestConfigDefaults` class's default behavior should correctly omit tests from coverage
- The configuration of the TestConfigDefaults instance should reflect its default behavior

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_integration_gate.py::TestConfigDefaults::test_provider_default_none 1ms 🛡️ 3

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' when it should be.

Key Assertions:

- The `config.provider` attribute of the test configuration object is set to 'none'.
- The `provider` attribute of the default configuration object is set to 'none'.
- The `get_default_config()` function returns a configuration object with a `provider` 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/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_integration_gate.py::TestConfigDefaults::test_secret_exclude_globs` 1ms 🛡️ 3

AI ASSESSMENT

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

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

Key Assertions:

- The `llm_context_exclude_globs` configuration setting is set to exclude 'secret' files.
- The `llm_context_exclude_globs` configuration setting is set to exclude '.env' files.
- Any secret files found in the excluded list are not included in the LLM context.
- No sensitive information like 'secret' or '.env' is present in the excluded list.
- The test ensures that only non-sensitive files are used for the 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: 387-388, 391, 395-397)`

PASSED `tests/test_integration_gate.py::TestFullPipeline::test_deterministic_output` 7ms 🛡️ 5

AI ASSESSMENT

Scenario: Verifies that the `test_deterministic_output` test reports are deterministic (sorted by nodeid).

Why Needed: This test prevents regression and ensures that the output of the pipeline is always in a predictable order.

Key Assertions:

- The `nodeids` list should be sorted in ascending order.
- The `nodeids` list should contain only unique values.
- All node IDs should be present in the sorted list.
- No duplicates should be found in the sorted list.
- The sorting is stable (i.e., if two nodes have the same ID, their original order is preserved).
- The sorting is consistent across different runs of the test.
- The `nodeids` list should not contain any empty 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/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: 387-388, 391, 395-397)
`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)

PASSED `tests/test_integration_gate.py::TestFullPipeline::test_empty_test_suite` 6ms 🛡️ 5

AI ASSESSMENT

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

Why Needed: This test prevents regression where an empty test suite is expected to produce a valid report, but the current implementation does not handle this case correctly.

Key Assertions:

- The total count of tests in the report should be zero.
- The summary section of the report should have a 'total' key with a value of zero.
- All test data loaded from the report.json file should be empty.
- The report writer does not throw an error when writing to an empty report.
- The report writer correctly handles an empty test suite by producing a valid report.
- The report summary is correct and does not contain any invalid values.
- The report data is correct and does not contain any missing or invalid 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: 387-388, 391, 395-397)
`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)

PASSED tests/test_integration_gate.py::TestFullPipeline::test_html_report_generation 32ms 🛡️ 5

AI ASSESSMENT

Scenario: Verifies that the full pipeline generates an HTML report.

Why Needed: Prevents a potential regression where the HTML report is not generated correctly or does not contain expected information.

Key Assertions:

- The test passes and creates an HTML file at the specified path.
- The HTML file contains the string '' in its content.
- The HTML file contains the string 'test_pass' in its content.
- The report is created with a valid configuration that specifies the report HTML file.
- The report HTML file exists at the expected location.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
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)

PASSED tests/test_integration_gate.py::TestFullPipeline::test_json_report_generation 55ms 🛡️ 7

AI ASSESSMENT

Scenario: The test verifies that a full pipeline generates a valid JSON report with the correct schema version, summary statistics, and number of tests.

Why Needed: This test prevents regression in the integration gate where the JSON report is generated. Without this test, the pipeline may produce incorrect or incomplete reports, leading to issues downstream.

Key Assertions:

- data['schema_version'] == SCHEMA_VERSION
- data['summary']['total'] == 3
- data['summary']['passed'] == 1
- data['summary']['failed'] == 1
- data['summary']['skipped'] == 1

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: 387-388, 391, 395-397)
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 object has all required fields.

Why Needed: This test prevents a potential bug where the report root is missing 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: 387-388, 391, 395-397)

PASSED `tests/test_integration_gate.py::TestSchemaCompatibility::test_run_meta_has_aggregation_fields` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'RunMeta has aggregation fields' verifies that the RunMeta object has 'is_aggregated', 'run_count', and possibly other aggregation policy fields.

Why Needed: This test prevents regression where a RunMeta object is created without any aggregation policies, potentially leading to incorrect results or errors in downstream processing.

Key Assertions:

- is_aggregated
- run_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	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: 387-388, 391, 395-397)

PASSED `tests/test_integration_gate.py::TestSchemaCompatibility::test_run_meta_has_status_fields` 1ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the RunMeta object is not populated with the necessary status fields, potentially leading to incorrect analysis results.

Key Assertions:

- The 'exit_code' field should be present in the data.
- The 'interrupted' field should be present in the data.
- The 'collect_only' field should be present in the data.
- The 'collected_count' field should be present in the data.
- The 'selected_count' 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	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: 387-388, 391, 395-397)

PASSED `tests/test_integration_gate.py::TestSchemaCompatibility::test_schema_version_defined` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Verifies that the schema version is defined and matches a semantic version.

Why Needed: Prevents a potential bug where the schema version is not defined or does not match a semantic version, potentially causing integration issues.

Key Assertions:

- SCHEMA_VERSION is set to a valid string.
- SCHEMA_VERSION contains at least one dot (.) character.
- SCHEMA_VERSION matches a semantic version format (e.g., '1.2.3'),

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: 387-388, 391, 395-397)`

PASSED `tests/test_integration_gate.py::TestSchemaCompatibility::test_test_case_has_required_fields` 1ms 🛡️ 3

AI ASSESSMENT

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

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

Key Assertions:

- The 'nodeid' key should be present in the `data` dictionary.
- The 'outcome' key should be present in the `data` dictionary.
- The 'duration' key 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: 387-388, 391, 395-397)

PASSED `tests/test_llm.py::TestGetProvider::test_gemini_returns_provider` 1ms 🛡️ 5

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 bug where the provider returned is not correctly identified as 'GeminiProvider' due to incorrect configuration or implementation.

Key Assertions:

- The `get_provider` function should return an instance of `GeminiProvider` when `provider='gemini'`
- The `provider` parameter should be set to `'gemini'` for the test to pass
- The returned provider instance should have a class name matching 'GeminiProvider'
- The `get_provider` function should correctly identify the provider as 'GeminiProvider' in this 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/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: 387-388, 391, 395-397)

PASSED `tests/test_llm.py::TestGetProvider::test_litellm_returns_provider` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the `get_provider` function returns an instance of LiteLLMProvider when a provider with name 'litellm' is specified.

Why Needed: This test prevents a potential bug where the `get_provider` function does not return an instance of LiteLLMProvider if the provided provider is not recognized ('litellm').

Key Assertions:

- The `provider` attribute of the returned `LiteLLMProvider` instance should be set to 'liteLLM'.
- The `__class__.__name__` attribute of the `LiteLLMProvider` instance should match 'LiteLLMProvider'.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_llm.py::TestGetProvider::test_none_returns_noop 1ms 🛡️ 5

AI ASSESSMENT

Scenario: test_get_provider_with_none_as_provider returns NoopProvider.

Why Needed: This test prevents a regression where the LLM is not properly initialized with a non-existent provider.

Key Assertions:

- The `get_provider` function should return an instance of `NoopProvider` when provided with a 'none' configuration.
- The `provider` attribute of the returned `NoopProvider` instance should be set to `'none'`.
- The `isinstance(provider, NoopProvider)` assertion should pass for the `NoopProvider` instance returned by `get_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/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: 387-388, 391, 395-397)

PASSED `tests/test_llm.py::TestGetProvider::test_ollama_returns_provider` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that OllamaProvider is returned when the `provider` parameter is set to 'ollama' in the provided configuration.

Why Needed: This test prevents a potential bug where the correct provider type is not detected if the `httpx` library is missing or not properly installed.

Key Assertions:

- The `get_provider()` function should return an instance of OllamaProvider.
- The `provider` parameter in the configuration object should be set to 'ollama'.
- The `__class__.__name__` attribute of the returned provider instance should match 'OllamaProvider'.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_llm.py::TestGetProvider::test_unknown_raises 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that an unknown provider raises a ValueError when trying to get a provider.

Why Needed: This test prevents the regression of not raising a ValueError for unknown providers.

Key Assertions:

- The function `get_provider` should raise a `ValueError` with message 'unknown' when called with an unknown provider.
- The error message should contain the string 'unknown'.
- When an unknown provider is passed to `get_provider`, it should not be able to return any value.
- The function call should fail and raise an exception instead of returning a result.
- The test should pass if the function raises a ValueError with the specified error message.
- The test should fail if the function does not raise a ValueError with the specified error message.
- The test should only pass if the `Config` class is correctly configured to handle unknown providers.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm.py::TestLlmProviderContract::test_noop_implements_interface` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Test that NoopProvider implements LlmProvider interface.

Why Needed: Prevents a potential bug where the NoopProvider class does not implement all required methods of the LlmProvider contract.

Key Assertions:

- provider should have annotate method
- provider should have is_available method
- provider should have get_model_name method
- provider should have config attribute

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm.py::TestNoopProvider::test_annotate_returns_empty` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the annotate method of NoopProvider returns an empty LlmAnnotation object when given a TestCaseResult.

Why Needed: This test prevents regression where the annotate method does not return any annotation when given a TestCaseResult with no scenario or why.

Key Assertions:

- annotation is an instance of LlmAnnotation
- annotation has an empty scenario
- annotation has an empty why_needed
- annotation has an empty key_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: 387-388, 391, 395-397)

PASSED `tests/test_llm.py::TestNoopProvider::test_get_model_name_empty` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The NoopProvider returns an empty string when the model name is not specified.

Why Needed: This test prevents a potential bug where the model name is not provided and the provider returns an empty string.

Key Assertions:

- assert provider.get_model_name() == ''
- assert provider.get_model_name() != 'noop' (to avoid false positives)
- assert config.model_name is None (to ensure no model name was specified)
- assert provider.get_model_name_from_config(config) == '' (to verify the default behavior)
- assert provider.get_model_name_from_str('noop') == '' (to test the edge case)
- assert provider.get_model_name_from_str('model_name') is None (to ensure no model name was 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/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: 387-388, 391, 395-397)

PASSED `tests/test_llm.py::TestNoopProvider::test_is_available` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The NoopProvider instance should always be available.

Why Needed: This test prevents a potential bug where the provider might not be available due to some internal issue.

Key Assertions:

- provider.is_available() == True
- provider._noop_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/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: 387-388, 391, 395-397)`

PASSED `tests/test_llm_annotator.py::test_annotate_tests_emits_summary` 1ms 🛡️ 6

AI ASSESSMENT

Scenario: Verify that annotation summary is printed when annotations run.

Why Needed: This test prevents regression where annotation summary is not printed.

Key Assertions:

- The function `get_provider` from `pytest_llm_report.llm.annotator` returns a `FakeProvider` instance.
- The `test_case` in the `TestCaseResult` object has an outcome of 'passed'.
- The captured output contains the string 'Annotated 1 test(s) via litellm'.

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: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

PASSED `tests/test_llm_annotator.py::test_annotate_tests_reports_progress` 1ms 🛡️ 6

AI ASSESSMENT

Scenario: Test that the progress report is generated for all test cases in the annotate_tests function.

Why Needed: This test prevents regression where the progress report is not generated for some test cases, potentially leading to incorrect results or missed tests.

Key Assertions:

- The test case should start with a message indicating that LLM annotations are being started.
- The test case should have a unique identifier and outcome.
- The progress messages should include the provider name (in this case, 'litellm') and the test case scenario.
- Each test case should be annotated with one or more messages that describe the annotation process.
- The number of annotations for each test case should match the expected number based on the test case outcome.
- The progress report should include a message indicating whether the LLM annotation is complete or not.

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: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

PASSED `tests/test_llm_annotator.py::test_annotate_tests_respects_opt_out_and_limit` 1ms 🛡️ 6

AI ASSESSMENT

Scenario: Tests for LLM annotations respecting opt-out and limit settings.

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

Key Assertions:

- The `provider` function is called with a valid configuration.
- The `llm_annotation` attribute is accessed on the first test result.
- The second and third test results have their `llm_annotation` attributes set to `None`.
- The `provider.calls` list contains only one call to `get_provider` with the correct configuration.
- All three test results have valid `outcome` values (passed).

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: 387-388, 391, 395-397)
`src/pytest_llm_report/prompts.py`	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

PASSED `tests/test_llm_annotator.py::test_annotate_tests_respects_rate_limit` 1ms 🛡️ 6

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the annotator does not respect the rate limit and makes excessive calls to the provider.

Key Assertions:

- The correct list of nodes should be ['tests/test_a.py::test_a', 'tests/test_b.py::test_b']
- The sleep function call should have been made at minute 2.0, not immediately.

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: 387-388, 391, 395-397)
`src/pytest_llm_report/prompts.py`	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

PASSED `tests/test_llm_annotator.py::test_annotate_tests_skips_unavailable_provider` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that annotation with unavailable providers skips the test.

Why Needed: To prevent regression when an unavailable LLM provider is used for annotation.

Key Assertions:

- The `is_available` method of the `UnavailableProvider` class returns False.
- The `get_provider` function from `pytest_llm_report.llm.annotator` calls the `UnavailableProvider` instance with a valid configuration.
- The `llm.annotator.get_provider` call is made before checking if the provider is available.
- The annotation process skips the test when an unavailable provider is used.
- The message 'is not available' is printed to the console during the annotation process.
- The `tmp_path` and `config` variables are preserved between 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/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: 387-388, 391, 395-397)

PASSED `tests/test_llm_annotator.py::test_annotate_tests_uses_cache` 1ms 🛡️ 6

AI ASSESSMENT

Scenario: Tests that annotations are cached between runs and that the annotation function is called when it should be.

Why Needed: This test prevents potential regression where the annotation function is not called even though the tests have passed.

Key Assertions:

- The `provider.calls` assertion checks if the `get_provider` method of `pytest_llm_report.llm.annotator` was called with the correct configuration.
- The `test.llm_annotation` assertion checks if `test.llm_annotation` is not `None` after calling `annotate_tests` with the same configuration.
- The `test.llm_annotation.scenario` assertion checks if `test.llm_annotation`'s scenario matches 'cached'.
- The `provider_next.annotate` assertion checks if the annotation function was called when it should be, and raises an AssertionError otherwise.

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: 387-388, 391, 395-397)
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 🛡️ 2

AI ASSESSMENT

Scenario: Verify that the `test_required_fields` test verifies the presence of 'scenario' and 'why_needed' fields in the annotation.

Why Needed: This test prevents a potential bug where the schema is not correctly defined or used, potentially leading to incorrect validation or errors.

Key Assertions:

- The function checks if 'scenario' and 'why_needed' are present in the required list of annotations.
- If either 'scenario' or 'why_needed' is missing from the required list, it raises an AssertionError with a descriptive 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/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_llm_contract.py::TestAnnotationSchema::test_schema_from_dict` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the AnnotationSchema.from_dict method to ensure it correctly parses a dictionary.

Why Needed: This test prevents potential bugs in the AnnotationSchema class where incorrect or missing data is passed to its methods.

Key Assertions:

- The annotation schema's scenario attribute should be set to the provided value.
- The annotation schema's why_needed attribute should match the expected value.
- The number of key assertions in the annotation schema should be equal to the specified count.
- The confidence level of the annotation schema should be within the expected range (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: 387-388, 391, 395-397)

PASSED `tests/test_llm_contract.py::TestAnnotationSchema::test_schema_handles_empty` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the AnnotationSchema class correctly handles an empty input.

Why Needed: Prevents a potential bug where the AnnotationSchema class does not validate or handle empty inputs correctly.

Key Assertions:

- The schema should be able to parse and validate an empty dictionary.
- The schema should report an error when encountering an empty input.
- The schema should not silently ignore or skip invalid inputs.
- The schema should provide a clear indication of the expected output for an empty input.
- The schema should handle nested structures correctly when dealing with empty inputs.
- The schema should validate and return the correct expected output for an empty input.
- The schema's validation logic should be robust and handle edge cases 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/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_llm_contract.py::TestAnnotationSchema::test_schema_handles_partial` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the AnnotationSchema from_dict method correctly sets the scenario attribute to 'Partial only' when a partial input is provided.

Why Needed: This test prevents regression where the AnnotationSchema's behavior changes unexpectedly when encountering partial inputs.

Key Assertions:

- schema.scenario should be set to 'Partial only'
- schema.why_needed should not be empty
- schema.scenario 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/llm/schemas.py`	5 lines (ranges: 77-81)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_llm_contract.py::TestAnnotationSchema::test_schema_has_required_fields` 1ms 🛡️ 2

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the annotation schema is missing necessary field definitions.

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']
- assert isinstance(ANNOTATION_JSON_SCHEMA['properties']['key_assertions'], list)
- assert len(ANNOTATION_JSON_SCHEMA['properties']['key_assertions']) > 0
- assert all(isinstance(assertion, str) for assertion in ANTONALOGY_ASSERTIONS)
- assert all(isinstance(assertion, dict) for assertion in ANNOTATION_JSON_SCHEMA['properties']['key_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`	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_llm_contract.py::TestAnnotationSchema::test_schema_to_dict 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the serialization of AnnotationSchema to dict.

Why Needed: This test prevents a regression where the schema is not properly serialized to a dictionary.

Key Assertions:

- asserts that the 'scenario' key matches the expected value.
- asserts that the 'why_needed' key matches the expected value.
- asserts that the 'key_assertions' key exists in the data and its value is 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/schemas.py	7 lines (ranges: 90-92, 94-96, 98)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_llm_contract.py::TestNoopProvider::test_noop_from_factory 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Tests the Factory to return a NoopProvider when the provider is 'none'.

Why Needed: Prevents regression in case the factory returns an incorrect provider.

Key Assertions:

- The function `get_provider` should return a NoopProvider for the provider='none' configuration.
- The `isinstance(provider, NoopProvider)` assertion should pass when `provider` is indeed a NoopProvider.
- The `assert` statement with `self` as the first argument should not raise an AssertionError.
- The function `get_provider` should be able to handle different provider configurations correctly.
- The `Config` class should be able to create a valid configuration object for 'none' provider.
- The `provider` variable should hold the correct value after calling `get_provider`.
- The `assert isinstance(provider, NoopProvider)` assertion should pass without any 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/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: 387-388, 391, 395-397)

PASSED `tests/test_llm_contract.py::TestNoopProvider::test_noop_is_llm_provider` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The `NoopProvider` class should be a subclass of `LlmProvider`.

Why Needed: This test prevents a bug where the `NoopProvider` is not properly inherited from `LlmProvider`.

Key Assertions:

- provider is an instance of LlmProvider
- provider is a subclass of NoopProvider
- provider has a type hint of LlmProvider

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm_contract.py::TestNoopProvider::test_noop_returns_empty_annotation` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The NoopProvider should return an empty annotation when the test function does not modify any variables.

Why Needed: This test prevents a regression where the NoopProvider returns incorrect annotations for tests that do not modify any variables.

Key Assertions:

- assert result.scenario == "" (empty string)
- assert result.why_needed == "" (empty string)
- assert result.key_assertions == [] (no key assertions 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/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: 387-388, 391, 395-397)`

PASSED `tests/test_llm_contract.py::TestProviderContract::test_annotate_returns_annotation` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the `annotate` method returns a `TestCaseResult` object with the required attributes.

Why Needed: This test prevents regression where the `annotate` method does not return a valid `TestCaseResult` object.

Key Assertions:

- The `scenario` attribute is present and has the expected value.
- The `why_needed` attribute is present and has the expected value.
- The `key_assertions` list contains all critical checks performed 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	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: 387-388, 391, 395-397)

PASSED `tests/test_llm_contract.py::TestProviderContract::test_provider_handles_empty_code` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the `provider` correctly handles an empty code block.

Why Needed: This test prevents a potential bug where an empty code block would cause the contract to fail.

Key Assertions:

- The `test_result` object is not None after calling `provider.annotate()`.
- The `nodeid` attribute of the `test_result` object matches the expected value.
- The `outcome` attribute of the `test_result` object is set to 'passed'.
- The `code` attribute of the `test_result` object contains an empty string.
- The `annotations` dictionary returned by `provider.annotate()` does not contain any errors or warnings.
- The `result` object passed to `provider.annotate()` is a valid `TestCaseResult` instance.
- The `config` object passed to `provider` has the correct `nodeid` and `outcome` 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/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: 387-388, 391, 395-397)

PASSED `tests/test_llm_contract.py::TestProviderContract::test_provider_handles_none_context` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the `provider` handles a `None` context gracefully by annotating a `TestCaseResult` with `code`.

Why Needed: This test prevents a potential bug where the annotation of a `TestCaseResult` with `code` would fail due to the absence of a value for the `code` field when provided with a `None` context.

Key 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: 387-388, 391, 395-397)

PASSED `tests/test_llm_contract.py::TestProviderContract::test_provider_has_annotate_method` 1ms 🛡️ 6

AI ASSESSMENT

Scenario: Test that all providers have an 'annotate' method.

Why Needed: Prevents regression in the contract where a provider might not have this method.

Key Assertions:

- The provider has an attribute named 'annotate'.
- The provider is callable.
- The provider has an 'annotate' method with no arguments.
- The provider does not raise any exceptions when calling its 'annotate' method.
- The provider's 'annotate' method returns a value (e.g., string, list).
- The provider's 'annotate' method calls itself recursively without terminating.
- The provider's 'annotate' method has the correct signature (method name and parameters).

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED `tests/test_llm_providers.py::TestGeminiProvider::test_annotate_handles_context_too_large` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The `annotate` method of the `GeminiProvider` class is called with a context that is too large, causing an error.

Why Needed: This test prevents a potential bug where the `annotate` method throws an exception when handling contexts larger than expected.

Key Assertions:

- The `context` parameter passed to `annotate` should not be longer than 1024 bytes.
- The `context` parameter passed to `annotate` should contain only strings and dictionaries.
- The `annotate` method should raise an exception when the context is too large.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestGeminiProvider::test_annotate_missing_dependency` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The LiteLLMProvider should correctly report a missing dependency when the required package is not installed.

Why Needed: This test prevents a potential bug where the provider incorrectly reports a missing dependency, causing users to install the wrong package instead of using LiteLLM.

Key Assertions:

- annotation.error == 'litellm not installed. Install with: pip install litellm'
- provider.annotate(...) returns an annotation object with error message
- assert True is asserted in test_case()
- test_case() contains a comment indicating that the required package is missing
- mock_import_error('litellm') is called to simulate a missing dependency
- config.provider == 'litellm' is set correctly
- LiteLLMProvider(...) is instantiated with correct configuration
- CaseResult(...) is created with correct nodeid and outcome

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestGeminiProvider::test_annotate_missing_token 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Test that annotating a missing API token prevents the error GEMINI_API_TOKEN is not set.

Why Needed: This test prevents a bug where the LLM provider throws an error when it cannot find the required API token.

Key Assertions:

- The annotation should fail with the message 'GEMINI_API_TOKEN is not set'.
- The annotation should have the correct provider and nodeid.
- The annotation should have a valid error message.
- The annotation should be able to identify the missing API token as the cause of the failure.
- The annotation should provide a meaningful error message that indicates the root cause of the issue.
- The annotation should not throw an error when the API token is present, but the provider still requires it.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestGeminiProvider::test_annotate_records_tokens` 1ms 🛡️ 6

AI ASSESSMENT

Scenario: Verify that tokens are recorded correctly on the limiter.

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

Key Assertions:

- The 'status' of the response should be ok.
- The 'totalTokenCount' in the response payload should match the expected value.
- The 'candidates' list in the response payload should contain a single dictionary with a 'text' key.
- The 'usageMetadata' dictionary in the response payload should have a 'totalTokenCount' key and its value should be 123.
- The rate limits logic should run without 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`	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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestGeminiProvider::test_annotate_retries_on_rate_limit` 1ms 🛡️ 6

AI ASSESSMENT

Scenario: The `test_annotate_retries_on_rate_limit` test verifies that the LLM provider annotates retries on rate limits.

Why Needed: This test prevents a regression where the LLM provider does not annotate retries on rate limits, potentially causing incorrect results or errors.

Key Assertions:

- The `rate_limit` attribute of the LLM provider is accessed before retrying.
- The `retry_on_rate_limit` method is called with the correct arguments (e.g., `max_retries`, `retries_delay`).
- The `annotate` method is called on the LLM provider with the correct arguments (e.g., `rate_limit`, `retry_on_rate_limit`).
- The `retry` method is called on the LLM provider with the correct arguments (e.g., `max_retries`, `retries_delay`).
- The `time.sleep` function is used to introduce a delay between retries.
- The `LLMProvider` instance has an attribute indicating that it supports rate limiting.
- The `rate_limit` and `retry_on_rate_limit` attributes are set correctly on the LLM provider instance before retrying.
- The `annotate` method returns a boolean value indicating whether the annotation was successful.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestGeminiProvider::test_annotate_rotates_models_on_daily_limit 1ms 🛡️ 6

AI ASSESSMENT

Scenario: The test verifies that the `annotate` method rotates models on the daily limit.

Why Needed: This test prevents a potential bug where the model rotation is not applied correctly when the daily limit is exceeded.

Key Assertions:

- The `rotate_models_on_daily_limit` method should rotate all models to the next day if the current date exceeds the daily limit.
- Each model should have its `id` attribute updated to reflect the new date.
- The `annotate` method should update the `metadata` dictionary with the rotated model information.
- All models should be marked as 'rotated' in the `models` list.
- The `rotate_models_on_daily_limit` method should not rotate models that are already on the next day (i.e., the current date is greater than or equal to the daily limit).
- If a model is rotated, its `metadata` dictionary should contain the correct information about the rotation (e.g., the new date and any relevant metadata).

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestGeminiProvider::test_annotate_skips_on_daily_limit` 1ms 🛡️ 6

AI ASSESSMENT

Scenario: The test verifies that the annotate method skips daily limit checks when it is called.

Why Needed: This test prevents a potential regression where the annotation process might skip daily limits due to performance or resource constraints.

Key Assertions:

- monkeypatch.assert_called_once_with(monkeypatch)
- self.annotate_skips_on_daily_limit.called_once_with(self, 'daily_limit')
- self.annotate_skips_on_daily_limit.return_value == False
- self.annotate_skips_on_daily_limit.return_value is not None
- self.annotate_skips_on_daily_limit.__name__ == 'skips_on_daily_limit'
- self.annotate_skips_on_daily_limit.__doc__ == 'Skips daily limit checks when annotating'
- self.annotate_skips_on_daily_limit.__annotations__ == {'skips_on_daily_limit': bool}
- self.annotate_skips_on_daily_limit.__defaults__ == (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`	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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestGeminiProvider::test_annotate_success_with_mock_response 1ms 🛡️ 6

AI ASSESSMENT

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

Why Needed: Prevents regressions by ensuring the correct annotation of a successful response from the LiteLLM provider.

Key Assertions:

- The annotation contains the correct scenario 'Checks login'.
- The annotation contains the correct why needed 'Stops regressions'.
- The annotation contains the correct key assertions ['status ok', 'redirect'].
- The annotation has a high confidence level of 0.8.
- The captured model is set to 'gpt-4o'.
- The test login function is found in the captured messages.
- The def test_login() function is also found in the captured 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/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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestGeminiProvider::test_exhausted_model_recovers_after_24h 1ms 🛡️ 6

AI ASSESSMENT

Scenario: The LLM model recovers after 24 hours of being exhausted.

Why Needed: This test prevents a regression where the model does not recover from exhaustion and returns incorrect results.

Key Assertions:

- The model's output should be similar to its original value within 24 hours.
- The model's output should decrease in magnitude over time, indicating recovery.
- The model's output should remain constant or increase slowly after 24 hours.
- The model's error rate should not increase significantly over the first 24 hours.
- The model's latency should be similar to its original value within 24 hours.
- The model's memory usage should decrease over time, indicating recovery.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestGeminiProvider::test_fetch_available_models_error 1ms 🛡️ 5

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the `fetch_available_models` method returns an error due to insufficient model availability.

Key Assertions:

- assertRaisesError with type 'RuntimeWarning' or 'ValueError'
- the `GeminiProvider.fetch_available_models()` method raises an error
- the error message is not a standard warning or ValueError
- the error message includes the string 'no available models'
- the error message includes the string 'insufficient data'
- the error message does not include any specific model names

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestGeminiProvider::test_model_list_refreshes_after_interval 1ms 🛡️ 6

AI ASSESSMENT

Scenario: The model list is refreshed after an interval of time.

Why Needed: This test prevents a potential regression where the model list does not refresh after an interval.

Key Assertions:

- The `refresh_interval` attribute of the provider is set to a valid number.
- The `model_list` attribute is updated with the latest models within the specified time frame.
- The `refreshed_at` timestamp for each model in the `model_list` is accurate and up-to-date.
- No stale or outdated models are present in the `model_list` after the interval has passed.
- The provider's behavior remains consistent across different 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/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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestLiteLLMProvider::test_annotate_handles_completion_error` 6.00s 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the LiteLLMProvider annotates completion errors correctly.

Why Needed: This test prevents a regression where the LLM provider does not surface completion errors in annotations.

Key Assertions:

- The 'boom' error is present in the annotation.
- The 'boom' error is reported as an error.
- The 'boom' error is included in the annotation's 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	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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestLiteLLMProvider::test_annotate_invalid_key_assertions 6.00s 🛡️ 6

AI ASSESSMENT

Scenario: Test that LiteLLMProvider rejects invalid key_assertions payloads.

Why Needed: To prevent the LiteLLMProvider from incorrectly handling invalid key_assertions payloads, which could lead to unexpected behavior or errors.

Key Assertions:

- Invalid response: key_assertions must be a list
- Key assertion error message should be informative and specific to the 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/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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestLiteLLMProvider::test_annotate_missing_dependency` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The LiteLLMProvider should report an error when a missing dependency is encountered.

Why Needed: This test prevents the LiteLLMProvider from silently failing when a required library is not installed, instead reporting the issue publicly.

Key Assertions:

- The annotation returned by `provider.annotate(test, 'def test_case(): assert True')` contains an error message that includes the name of the missing dependency (litellm).
- The error message should include a clear indication of what was installed and how to install it.
- The error message should be concise and easy to understand for users who may not be familiar with pip or package management.
- The test should fail if the mock import error is not raised when a missing dependency is encountered.
- The test should pass if the mock import error is raised correctly, indicating that an error occurred.
- The annotation returned by `provider.annotate(test, 'def test_case(): assert True')` should include the name of the missing dependency (litellm) in its message.
- The message should not contain any misleading information about how to install the required library.
- The test should only fail if the mock import error is raised when a missing dependency is encountered, and pass otherwise.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestLiteLLMProvider::test_annotate_success_with_mock_response` 1ms 🛡️ 6

AI ASSESSMENT

Scenario: Test that LiteLLMProvider annotates a successful response with the expected key assertions and confidence level.

Why Needed: This test prevents regressions by ensuring that LiteLLMProvider correctly annotates responses from the liteellm module.

Key Assertions:

- status ok
- redirect
- confidence >= 0.8
- model = 'gpt-4o'
- messages[0]['role'] == 'system'
- tests/test_auth.py::test_login in messages[1]['content']

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestLiteLLMProvider::test_is_available_with_module 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Test that the LiteLLM provider detects installed modules correctly.

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

Key Assertions:

- The `is_available()` method of the `LiteLLMProvider` class returns True when the 'litellm' module is available in the system's modules.
- The `is_available()` method should return False if the 'litellm' module is not installed or not found in the system's modules.
- The test should fail if the 'litellm' module is not available in the system's modules, indicating a bug in the provider.
- The test should pass when the 'litellm' module is installed and present in the system's modules.
- The test should also pass when the 'litellm' module is not installed or not found in the system's modules, but the provider still detects it 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	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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestOllamaProvider::test_annotate_fallbacks_on_context_length_error 1ms 🛡️ 6

AI ASSESSMENT

Scenario: Tests the functionality of annotating fallbacks when context length is an error.

Why Needed: Prevents a regression where annotating fallbacks fails due to context length errors in certain scenarios.

Key Assertions:

- A fallback annotation should be added to the context when it exceeds the maximum allowed length.
- The fallback annotation should have the correct type (e.g., 'OllamaProvider') and key.
- The fallback annotation should not be removed or hidden after being annotated.
- The fallback annotation should only be added if the original annotation is missing or empty.
- The fallback annotation should not interfere with other annotations in the context.
- The fallback annotation should be preserved when switching between different LLM providers.
- The fallback annotation should be correctly propagated to downstream consumers of the annotated 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/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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_annotate_handles_call_error` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the Ollama provider correctly annotates a call to `test_case` with an error message when it fails due to a system prompt.

Why Needed: This test prevents regression in handling call errors, ensuring the annotation is accurate and informative even when the system prompt returns an unexpected value.

Key Assertions:

- The annotation should include the 'Failed after 3 retries. Last error: boom' message as expected.
- The annotation should not include any additional information other than the 'Failed after 3 retries. Last error: boom' message.
- The annotation should only include the 'Failed after 3 retries. Last error: boom' message, without any other details about the system prompt.
- The annotation should not raise an exception when it fails to annotate the call (e.g., due to a timeout or network issue).
- The annotation should be able to handle different types of system prompts (e.g., 'boom', 'error', etc.).

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_annotate_missing_httpx` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Ollama provider reports missing httpx dependency.

Why Needed: The test prevents a bug where the Ollama provider incorrectly assumes that httpx is installed when it's not.

Key Assertions:

- assert annotation.error == 'httpx not installed. Install with: pip install httpx'
- assert annotation.nodeid == 'tests/test_sample.py::test_case'
- assert annotation.outcome == 'passed'

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_annotate_success_full_flow` 1ms 🛡️ 6

AI ASSESSMENT

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

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

Key Assertions:

- Check if the status code is 200 (OK) and the response contains a JSON object with expected keys
- Validate the presence of a valid token in the 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	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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_call_ollama_success` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Test that the Ollama provider makes a correct API call to generate text.

Why Needed: Prevents regression in case of incorrect or missing timeout settings.

Key Assertions:

- The response from the Ollama model is 'test response'.
- The URL used for the API call is 'http://localhost:11434/api/generate'.
- The model used by the Ollama provider is 'llama3.2:1b'.
- The prompt used to generate text is 'test prompt'.
- The system prompt used to generate text is 'system prompt'.
- The stream flag for the generated text is False.
- The timeout setting for the API call is 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/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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_call_ollama_uses_default_model` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Ollama provider uses default model when not specified.

Why Needed: To prevent a regression where the Ollama provider fails to use the default model if it is not provided in the configuration.

Key Assertions:

- The 'model' key in the captured dictionary should be equal to 'llama3.2'.
- The 'json' value in the captured dictionary should contain a 'response' key with the value 'ok'.
- The 'model' key in the captured dictionary should not be empty.
- The 'json' value in the captured dictionary should not be None.
- The 'response' key in the captured dictionary should have a 'response' key with the value 'ok'.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_check_availability_failure` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Verify 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 `_check_availability()` method of the `OllamaProvider` instance should return `False` when the server is unavailable.
- The `config` object passed to the `OllamaProvider` constructor has a valid `provider` set to 'ollama'.
- When the `fake_get()` function raises a `ConnectionError`, it should be caught and propagated as 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	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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_check_availability_non_200` 1ms 🛡️ 5

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the Ollama provider incorrectly reports availability for non-200 status codes, causing downstream services to rely on incorrect information.

Key Assertions:

- The function _check_availability() returns False for any URL that has a status code other than 200.
- The function _check_availability() does not raise an exception when encountering a non-200 status code.
- The provider's method to check availability is called with the correct argument (status_code) even if it's not 200.
- The provider's method to check availability returns False for any URL that has a status code other than 200.
- The provider's method to check availability does not raise an exception when encountering a non-200 status code.
- The configuration provider is set correctly and the OllamaProvider instance is created with it.
- The config provider is set correctly and the provider instance is created with it.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_check_availability_success` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Verifies that the Ollama provider checks availability via /api/tags endpoint successfully.

Why Needed: This test prevents a potential bug where the provider does not return an error or message when it cannot find the tags, potentially leading to unexpected behavior in downstream applications.

Key Assertions:

- The '/api/tags' URL is present in the provided URL.
- The response status code is 200 (OK).
- The 'tags' key is not present in the response. This indicates that the provider cannot find the tags.
- An error or message indicating failure to find the tags is returned by the provider.
- The provider raises an exception when it encounters a non-existent resource, such as a 404 Not Found 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`	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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_is_local_returns_true` 1ms 🛡️ 5

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the provider might not correctly identify if it's local or not.

Key Assertions:

- provider.is_local() == True
- provider.is_local() != False
- provider.is_local() is not None
- config.provider == 'ollama'
- OllamaProvider(config).is_local() is True
- OllamaProvider(config) 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	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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestOllamaProvider::test_parse_response_invalid_json 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the `OllamaProvider` class throws an error when attempting to parse a non-JSON response.

Why Needed: This test prevents a potential bug where the Ollama provider incorrectly interprets responses without proper validation.

Key Assertions:

- If the provided response is not JSON, the `_parse_response` method of `OllamaProvider` should throw an exception with the message 'Failed to parse LLM response as JSON'.
- The error message should include the string 'Failed to parse LLM response as JSON'.
- The test should fail if the provided response is not a valid JSON string.
- If the response contains any non-JSON characters, it should be considered invalid and trigger the exception.
- The `Config` class should be able to validate the input response against the expected format.
- The `_parse_response` method of `OllamaProvider` should raise an exception with the specified error message when encountering a non-JSON 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	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: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestOllamaProvider::test_parse_response_invalid_key_assertions 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Ollama provider rejects invalid key_assertions payloads when parsing responses.

Why Needed: This test prevents the Ollama provider from incorrectly handling invalid key_assertions in its responses.

Key Assertions:

- The response must be a list
- Key assertions should not contain any keys
- Invalid or missing keys are not allowed

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_parse_response_json_in_code_fence` 1ms 🛡️ 5

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 potential bugs where the provider incorrectly or incompletely extracts JSON from markdown code fences, potentially leading to incorrect or incomplete model training data.

Key Assertions:

- The extracted JSON is in the correct format and does not contain any invalid characters.
- The extracted JSON contains only valid JSON syntax and does not include any extraneous whitespace or formatting.
- The extracted JSON does not contain any malicious or unexpected data, such as unquoted strings or arrays with non-string values.
- The extracted JSON is a single object or array, rather than multiple objects or arrays.
- The extracted JSON contains only one top-level key-value pair, rather than multiple pairs.
- The extracted JSON does not contain any nested objects or arrays that exceed the maximum allowed depth.
- The extracted JSON does not contain any circular references or other self-referential data structures.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_parse_response_json_in_plain_fence` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: The provided test verifies that the Ollama provider correctly extracts JSON from a plain markdown fence without any language specification.

Why Needed: This test prevents regression in the case where the input JSON is not properly formatted or does not contain any valid keys.

Key Assertions:

- The response should be a JSON object with no properties (i.e., an empty dictionary).
- The response should only contain string values (e.g., 'hello').
- There should be no nested objects or arrays in the response.
- All string values should have a length of 1 character.
- No keys should be present in the JSON object.
- The JSON object should not have any circular references.
- The response should only contain strings and numbers, without any other types (e.g., booleans).

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_parse_response_success` 1ms 🛡️ 5

AI ASSESSMENT

Scenario: Test the Ollama provider's ability to parse valid JSON responses.

Why Needed: To prevent bugs in the Ollama provider that may occur when parsing invalid or malformed JSON responses.

Key Assertions:

- assert a is not None
- assert b 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	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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestArtifactEntry::test_to_dict` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that `CoverageEntry.to_dict()` correctly serializes the object.

Why Needed: Prevents regression in coverage entry serialization.

Key Assertions:

- The 'file_path' key is set to the expected value.
- The 'line_ranges' key is set to the expected value.
- The 'line_count' key is set to the expected value.
- The 'coverage_data' object is not created with an empty dictionary.
- The 'coverage_data' object has a non-empty dictionary for 'file_path'.
- The 'coverage_data' object has a non-empty dictionary for 'line_ranges'.
- The 'coverage_data' object has a non-empty dictionary for '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`	4 lines (ranges: 254-257)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestCollectionError::test_to_dict` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Testing the `to_dict` method of `CoverageEntry` class.

Why Needed: This test prevents a potential bug where the `to_dict` method does not correctly serialize the coverage entry data.

Key Assertions:

- The expected file path is 'src/foo.py'.
- The expected line ranges are '1-3, 5, 10-15'.
- The expected line count is 10.
- The `to_dict` method should return a dictionary with the specified keys and values.
- The `to_dict` method should not raise an exception when the coverage entry data is invalid or missing required information.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestCoverageEntry::test_to_dict` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests CoverageEntry to_dict method.

Why Needed: This test prevents a potential bug where the coverage entry serialization is incorrect.

Key Assertions:

- The 'file_path' key should be set to the actual file path.
- The 'line_ranges' key should contain valid range notation (e.g., '1-3, 5, 10-15').
- The 'line_count' key should match the expected value of 10.
- Any additional line ranges should be properly formatted and not exceed the maximum allowed length.
- If an invalid file path is provided, the test should fail with a clear error message.
- If any other unexpected issue occurs during serialization, the test should also fail 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/models.py`	4 lines (ranges: 40-43)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestLlmAnnotation::test_empty_annotation` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: An empty annotation should be created with default values.

Why Needed: This test prevents a regression where an empty annotation does not have default values.

Key Assertions:

- annotation.scenario = ''
- annotation.why_needed = ''
- annotation.key_assertions = []
- assert annotation.confidence is None
- assert annotation.error 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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestLlmAnnotation::test_to_dict_minimal` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `LlmAnnotation` object can be serialized to a dictionary with the required fields.

Why Needed: This test prevents regression by ensuring that the minimal annotation format includes all necessary keys for serialization.

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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestLlmAnnotation::test_to_dict_with_all_fields` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test to dictionary with all fields

Why Needed: Prevents auth bypass by ensuring a full annotation is created.

Key Assertions:

- Assert that the 'scenario' field matches the expected value.
- Assert that the 'confidence' field matches the expected value (0.95).
- Assert that the 'context_summary' field contains the expected values for mode and bytes.
- Verify that the dictionary is created with all required fields.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestReportRoot::test_default_report` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test default report schema version and empty lists.

Why Needed: Prevents a potential bug where the default report is missing required information.

Key Assertions:

- The 'schema_version' key should be present in the report dictionary with value equal to SCHEMA_VERSION.
- The 'tests' key should be an empty list.
- The 'warnings' key should not be included in the report dictionary.
- The 'collection_errors' key should not be included in the report 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`	`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: 387-388, 391, 395-397)`

PASSED `tests/test_models.py::TestReportRoot::test_report_with_collection_errors` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test Report with Collection Errors should include them.

Why Needed: This test prevents a bug where the report does not include collection errors when they are present.

Key Assertions:

- The length of `collection_errors` in the report is 1.
- The value of `nodeid` in the first element of `collection_errors` is 'test_bad.py'.
- Each error in `collection_errors` has a valid `nodeid` and `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/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: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestReportRoot::test_report_with_warnings 1ms 🛡️ 3

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 `ReportWarning` object is created with the correct warning code and message.
- The length of the `warnings` list in the report dictionary is 1.
- The first warning in the `warnings` list has the correct warning code.
- The warning code 'W001' is present in the first warning.
- The warning message 'No coverage' is present in the first warning.
- The `ReportWarning` object has a non-empty `code` attribute.
- The `ReportWarning` object has a non-empty `message` attribute.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestReportRoot::test_tests_sorted_by_nodeid` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests should be sorted by nodeid in output.

Why Needed: This test prevents regression where the order of tests is not guaranteed to match their original nodeid.

Key Assertions:

- The list of nodeids returned matches the expected order.
- Each nodeid appears only once in the list.
- No duplicate nodeids are present in the list.
- All nodeids are present in the list.
- Nodeids without tests are not included in the list.
- Nodeids with multiple tests are sorted correctly.
- The test order is preserved across different runs of the same 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/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: 387-388, 391, 395-397)`

PASSED `tests/test_models.py::TestReportWarning::test_to_dict_with_detail` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test `test_to_dict_with_detail` verifies that the `to_dict()` method of `ReportWarning` returns a dictionary with the 'detail' key.

Why Needed: This test prevents a warning about missing coverage details in reports.

Key Assertions:

- The 'detail' key should be present in the returned dictionary.
- The value of the 'detail' key should match the provided path '/path/to/file'.
- The 'message' and 'code' keys are not included in the returned 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	6 lines (ranges: 229-231, 233-235)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestReportWarning::test_to_dict_without_detail` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test to dictionary without detail should exclude it.

Why Needed: This test prevents a warning that occurs when the 'to_dict' method is called on a ReportWarning object without providing any additional details.

Key Assertions:

- The 'code' key in the dictionary should be equal to 'W001'.
- The 'message' key in the dictionary should be equal to 'No coverage'.
- The 'detail' key in the dictionary should not exist.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestRunMeta::test_aggregation_fields_present` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that RunMeta has aggregation fields.

Why Needed: This test prevents a regression where the aggregation policy is not correctly applied to multiple runs.

Key Assertions:

- assert d['run_id'] == 'run-123'
- assert d['run_group_id'] == 'group-456'
- assert d['is_aggregated'] is True
- assert d['aggregation_policy'] == 'merge'
- assert d['run_count'] == 3
- assert len(d['source_reports']) == 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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestRunMeta::test_llm_fields_excluded_when_disabled` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test LLM fields are excluded when annotations not enabled.

Why Needed: This test prevents a regression where the LLM fields (llm_annotations_enabled, llm_provider, and llm_model) are included in the metadata even when annotations are disabled.

Key Assertions:

- The 'llm_annotations_enabled' key is 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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestRunMeta::test_llm_traceability_fields` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test LLM traceability fields are included when enabled.

Why Needed: This test prevents regression in the case where llm_traceability_fields is disabled.

Key Assertions:

- The value of llm_annotations_enabled should be True.
- llm_provider should be set to 'ollama'.
- llm_model should be set to 'llama3.2:1b'.
- llm_context_mode should be set to 'complete'.
- llm_annotations_count should be 10.
- llm_annotations_errors should be 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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestRunMeta::test_non_aggregated_excludes_source_reports` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that non-aggregated reports do not include source_reports.

Why Needed: Prevents regression where non-aggregated reports are incorrectly including source_reports.

Key Assertions:

- The 'source_reports' key should be absent from the report dictionary.
- The 'is_aggregated' value should be set to False for non-aggregated 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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestRunMeta::test_run_meta_to_dict_full` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test RunMeta to dict with all optional fields.

Why Needed: Prevents regression in case of missing plugin or repository git SHA.

Key Assertions:

- assert data['git_sha'] == 'abc1234',
- assert data['git_dirty'] is True,
- assert data['repo_version'] == '1.0.0',
- assert data['repo_git_sha'] == 'abc1234',
- assert data['repo_git_dirty'] is True,
- assert data['plugin_git_sha'] == 'def5678',
- assert data['plugin_git_dirty'] is False,
- assert len(data['source_reports']) == 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/models.py`	`49 lines (ranges: 277-279, 281-283, 364-380, 382-405, 407, 419)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_models.py::TestRunMeta::test_run_status_fields` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the 'RunMeta' class to ensure it includes required run status fields.

Why Needed: This test prevents a potential regression where the 'RunMeta' object is missing certain critical information about its execution status.

Key Assertions:

- The 'exit_code' field of the 'RunMeta' object should be equal to 1.
- The 'interrupted' field of the 'RunMeta' object should be True.
- The 'collect_only' field of the 'RunMeta' object should be True.
- The 'collected_count' field of the 'RunMeta' object should be equal to 10.
- The 'selected_count' field of the 'RunMeta' object should be equal to 8.
- The 'deselected_count' field of the 'RunMeta' object 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	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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestSchemaVersion::test_schema_version_format` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Verifies the schema version is formatted as a semver string.

Why Needed: Prevents regression where the schema version is not in semver format.

Key Assertions:

- The schema version should be split into three parts (e.g., '1.2.3')
- Each part of the schema version should be a digit
- The length of each part should be exactly 3 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestSchemaVersion::test_schema_version_in_report_root 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the `ReportRoot` class includes the schema version in its report root.

Why Needed: This test prevents a regression where the schema version is not included in the report root.

Key Assertions:

- The `schema_version` attribute of `ReportRoot` should be equal to `SCHEMA_VERSION`.
- The value of `schema_version` in the JSON representation of `ReportRoot` should also be equal to `SCHEMA_VERSION`.
- The `to_dict()` method of `ReportRoot` returns a dictionary with a key-value pair where the key is `'schema_version'` and the value is `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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestSourceCoverageEntry::test_to_dict` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test coverage entry serialization.

Why Needed: Prevents regression in coverage reporting when file paths or line ranges change.

Key Assertions:

- The 'file_path' key is correctly set to the expected value.
- The 'line_ranges' key is correctly set to the expected format.
- The 'line_count' key is correctly set to 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	8 lines (ranges: 71-78)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestSourceReport::test_to_dict_minimal` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `to_dict` method of `LlmAnnotation` returns a dictionary with required fields.

Why Needed: This test prevents a potential bug where the minimal annotation is missing some required fields.

Key Assertions:

- The presence of 'scenario' in the dictionary is expected.
- The presence of 'why_needed' in the dictionary is expected.
- The absence of 'confidence' in the dictionary is expected (since it's optional and None by default).

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED tests/test_models.py::TestSourceReport::test_to_dict_with_run_id 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'SourceReport with run_id should include it' verifies that the SourceReport object includes its 'run_id' attribute.

Why Needed: This test prevents a bug where the 'run_id' is not included in the dictionary representation of the SourceReport object, potentially causing issues when serializing or comparing the object.

Key Assertions:

- The 'run_id' key should be present in the dictionary representation of the SourceReport object.
- The value of the 'run_id' key should match the expected string value.
- The 'run_id' key should not be missing from the dictionary representation of the SourceReport 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/models.py	6 lines (ranges: 277-279, 281-283)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestSummary::test_to_dict` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `CoverageEntry` class correctly serializes a coverage entry into a dictionary.

Why Needed: This test prevents a potential bug where the serialization of coverage entries is incorrect, potentially leading to unexpected behavior or errors in downstream applications.

Key Assertions:

- assert d['file_path'] == 'src/foo.py'
- assert d['line_ranges'] == '1-3, 5, 10-15'
- assert d['line_count'] == 10
- The `CoverageEntry` class correctly handles line ranges with multiple occurrences.
- The `CoverageEntry` class correctly handles overlapping line ranges.
- The `CoverageEntry` class correctly handles missing line ranges.
- The `CoverageEntry` class correctly handles invalid line range formats (e.g., non-integer values).
- The `to_dict()` method returns a dictionary with the correct keys and values for a coverage entry.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED `tests/test_models.py::TestTestCaseResult::test_minimal_result` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: tests/test_models.py::TestTestCaseResult::test_minimal_result

Why Needed: This test prevents a regression where the minimal result is missing required fields.

Key Assertions:

- The 'nodeid' field should be present and match the expected value.
- The 'outcome' field should be present and match the expected value.
- The 'duration' field should be present and have a value of 0.0.
- The 'phase' field should be present and 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`	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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestTestCaseResult::test_result_with_coverage` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test verifies that the `result` dictionary includes a single 'coverage' key with a list of coverage entries.

Why Needed: This test prevents regression where the coverage information is missing or incorrect.

Key Assertions:

- The 'coverage' key should be present in the 'result' dictionary and contain a list of coverage entries.
- Each coverage entry should have a 'file_path' attribute set to the expected file path.
- Each coverage entry should have a 'line_ranges' attribute set to the expected line ranges (1-5).
- Each coverage entry should have a 'line_count' attribute set to the expected line count (5).
- The list of coverage entries should contain exactly one element.
- All file paths in the list should match the expected file path ('src/foo.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	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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestTestCaseResult::test_result_with_llm_opt_out` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test verifies that the `TestCaseResult` object includes a flag indicating LLM opt-out.

Why Needed: This test prevents regression by ensuring that the `TestCaseResult` object accurately reflects the presence of LLM opt-out.

Key Assertions:

- The value of `llm_opt_out` in the `d` dictionary is set to `True`.
- The `llm_opt_out` key exists in the `d` dictionary.
- The value of `llm_opt_out` is a boolean value (`True` or `False`).
- The presence of LLM opt-out is verified through the `to_dict()` method.
- The `TestCaseResult` object is converted to a dictionary before assertions are made.
- The dictionary contains all required keys (nodeid, outcome, and llm_opt_out).
- Assertions about the value of `llm_opt_out` are performed on the resulting dictionary.
- The test ensures that the `TestCaseResult` object accurately represents LLM opt-out.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestTestCaseResult::test_result_with_rerun` 1ms 🛡️ 3

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 is not properly updated with rerun information.

Key Assertions:

- The value of `rerun_count` should be equal to 2.
- The value of `final_outcome` should be 'passed'.
- The `result` object should have the `to_dict()` method called on it.
- The `rerun_count` and `final_outcome` keys should exist in the `d` dictionary.
- The `rerun_count` key should contain a value of 2.
- The `final_outcome` key should contain a value of 'passed'.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestTestCaseResult::test_result_without_rerun_excludes_fields 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test case 'test_result_without_rerun_excludes_fields' verifies that the 'result' dictionary does not contain 'rerun_count' and 'final_outcome' keys.

Why Needed: This test prevents regression where the 'rerun_count' and 'final_outcome' fields are included in the result dictionary when rerunning tests without re-running the same code.

Key Assertions:

- The 'result' dictionary should not contain 'rerun_count' key.
- The 'result' dictionary should not contain 'final_outcome' key.
- The 'result' dictionary should exclude 'rerun_count' and 'final_outcome' keys when passed without rerunning the test.
- The 'result' dictionary should be consistent with the expected output after excluding 'rerun_count' and 'final_outcome' fields.
- The 'result' dictionary should not contain any other fields that are specific to reruns or final outcomes.
- The 'result' dictionary should only contain keys relevant to the test case's functionality.
- The 'result' dictionary should be consistent with the expected output after excluding certain fields.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestConfig::test_default_values` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that default values are set correctly for the TestConfig class.

Why Needed: Prevents potential configuration errors and inconsistencies when creating a new TestConfig instance.

Key Assertions:

- The provider should be set to 'none'.
- The llm_context_mode should be set to 'minimal'.
- The llm_max_tests should be set to 0.
- The llm_max_retries should be set to 3.
- The llm_context_bytes should be set to 32000.
- The llm_context_file_limit should be set to 10.
- The llm_requests_per_minute should be set to 5.
- The llm_timeout_seconds should be set to 30.
- The include_phase should be set to 'run'.
- The aggregate_policy should be set to 'latest'.
- The is_llm_enabled() method 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/options.py`	`3 lines (ranges: 107, 147, 224)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_options.py::TestConfig::test_get_default_config` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `get_default_config` function returns a valid default configuration.

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

Key Assertions:

- cfg is an instance of `Config`.
- cfg.provider == "none".
- cfg.provider should be set to 'none' by the factory 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/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestConfig::test_is_llm_enabled` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the `is_llm_enabled` check for different providers.

Why Needed: Prevents regression in case of provider changes or updates to the LLM provider.

Key Assertions:

- The function should return False when the provider is set to 'none'.
- The function should return True when the provider is set to 'ollama'.
- The function should not return a value when the provider is set to an unknown or invalid 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`	3 lines (ranges: 107, 147, 224)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestConfig::test_validate_invalid_aggregate_policy` 1ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed after 3 retries. Last error: 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/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: 387-388, 391, 395-397)

PASSED tests/test_options.py::TestConfig::test_validate_invalid_context_mode 1ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed after 3 retries. Last error: 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/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: 387-388, 391, 395-397)

PASSED tests/test_options.py::TestConfig::test_validate_invalid_include_phase 1ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed after 3 retries. Last error: 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/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: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestConfig::test_validate_invalid_provider` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test validates configuration with an invalid provider.

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

Key Assertions:

- The test verifies that there is exactly one error message related to an invalid provider.
- The test asserts that the error message contains the string 'Invalid provider '
- The test checks for the presence of the specific error message in the list of errors.
- The test verifies that the error message is not empty or null.
- The test ensures that the error message does not contain any other invalid providers.
- The test validates that the configuration is still valid after encountering an invalid 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	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: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestConfig::test_validate_numeric_ranges` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test validation of numeric constraints for Config object.

Why Needed: Prevents a potential bug where the configuration is not validated correctly, potentially leading to unexpected behavior or errors.

Key Assertions:

- The 'llm_context_bytes' value must be at least 1000.
- The 'llm_max_tests' value must be 0 (no limit) or positive.
- The 'llm_requests_per_minute' value must be at least 1.
- The 'llm_timeout_seconds' value must be at least 1.
- The 'llm_max_retries' value must be 0 or positive.
- All numeric constraints (context_bytes, max_tests, requests_per_minute, timeout_seconds, max_retries) must be validated 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`	`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: 387-388, 391, 395-397)`

PASSED `tests/test_options.py::TestConfig::test_validate_valid_config` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Valid configuration is validated without any errors.

Why Needed: This test prevents potential bugs that could occur when validating an invalid configuration.

Key Assertions:

- A valid configuration object is created and assigned to the cfg variable.
- The validate() method of the Config class returns an empty list (i.e., no errors).
- No exceptions are raised during the validation process.
- The cfg object has no attributes that would cause a validation error.
- All required fields in the configuration are present and have valid values.
- The configuration is not empty or None.
- The configuration does not contain any invalid data (e.g., missing, malformed, etc.)
- The configuration conforms to all expected schema definitions.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED `tests/test_options.py::TestLoadConfig::test_load_aggregation_options` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test loads aggregation options with default values.

Why Needed: This test prevents a potential bug where the aggregation policy is not set correctly when loading configuration.

Key Assertions:

- The aggregate_dir attribute of the config object should be equal to 'aggr_dir'.
- The aggregate_policy attribute of the config object should be equal to 'merge'.
- The aggregate_run_id attribute of the config object should be equal to 'run-123'.
- The aggregate_group_id attribute of the config object should be equal 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: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestLoadConfig::test_load_config_invalid_int_ini` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test Load Config: Handling Invalid Integer Values in INI

Why Needed: Prevents test from crashing due to invalid integer values in INI files.

Key Assertions:

- The function `load_config` should not crash when encountering an invalid integer value in the INI file.
- The default value for `llm_max_retries` should be used instead of a garbage value.
- The test should pass even if the `getini` method raises an exception.
- The test should verify that the fallback value is correct (3 in this case).
- The function `load_config` should not raise an exception when encountering invalid integer 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	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: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestLoadConfig::test_load_coverage_source` 1ms 🛡️ 3

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 coverage source is not correctly loaded due to an incorrect value being passed to the `load_config` function.

Key Assertions:

- The `llm_coverage_source` option should be set to 'cov_dir' when loading the coverage source.
- The `load_config` function should be called with the correct `llm_coverage_source` option value.
- The `cfg.llm_coverage_source` attribute should have a value of 'cov_dir'.
- The `mock_pytest_config.option.llm_coverage_source` attribute should return 'cov_dir' when accessed.
- The `load_config` function should correctly load the coverage source from the provided 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`	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: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestLoadConfig::test_load_defaults` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the default provider and report HTML are correctly loaded when no options are provided.

Why Needed: This test prevents a potential regression where the default provider ('none') or report HTML ('None') are not loaded correctly without any configuration.

Key Assertions:

- cfg.provider == 'none'
- cfg.report_html 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/options.py	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)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestLoadConfig::test_load_from_cli_overrides_ini` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that CLI options override ini options.

Why Needed: Prevents regression where CLI overrides ini settings and report HTML is not generated correctly.

Key Assertions:

- ini_value should be set to 'cli_report.html' for llm_report_html option
- llm_requests_per_minute should be set to 100 for llm_requests_per_minute option

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_options.py::TestLoadConfig::test_load_from_cli_retries 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `llm_max_retries` option is correctly set to 9 when loading from CLI.

Why Needed: This test prevents a bug where the `llm_max_retries` option is not properly set for retries in the command-line interface.

Key Assertions:

- The value of `llm_max_retries` should be equal to 9.
- The `llm_max_retries` option should be correctly set even if it's overridden by a CLI argument.
- The test should fail when `llm_max_retries` is not set or is less than 1.
- The value of `llm_max_retries` should remain unchanged after the test is 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	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: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestLoadConfig::test_load_from_ini` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test loading values from ini options for load_config function.

Why Needed: Prevents a potential bug where the correct values are not loaded from ini files due to incorrect configuration.

Key Assertions:

- The 'provider' key in the config dictionary should be set to 'ollama'.
- The 'model' key in the config dictionary should be set to 'llama3'.
- The 'context_mode' key in the config dictionary should be set to 'balanced'.
- The 'requests_per_minute' key in the config dictionary should be set to 10.
- The 'max_retries' key in the config dictionary should be set to 5.
- The 'html' key in the config dictionary should be set to 'report.html'.
- The 'json' key in the config dictionary 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: 387-388, 391, 395-397)

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 aggregation policy is set to 'merge' without specifying an aggregate group or include history.

Key Assertions:

- The `aggregate_dir` attribute of the test configuration should be equal to '/reports'.
- The `aggregate_policy` attribute of the test configuration should be equal to 'merge'.
- The `aggregate_include_history` attribute of the test configuration should be 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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 in case of multiple report formats being used.

Key Assertions:

- The `report_html` attribute is set to 'report.html'.
- The `report_json` attribute is set to 'report.json'.
- The `report_pdf` attribute is set to '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: 387-388, 391, 395-397)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_capture_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `capture_failed_output` option is set to `True`.

Why Needed: This test prevents a bug where the captured output exceeds the maximum allowed characters (8000).

Key Assertions:

- config.capture_failed_output is True
- config.capture_output_max_chars is 8000

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_compliance_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `metadata_file` and `hmac_key_file` are set correctly in the test configuration.

Why Needed: This test prevents a bug where the compliance settings are not properly configured, potentially leading to errors or unexpected behavior.

Key Assertions:

- The `metadata_file` is set to 'metadata.json'.
- The `hmac_key_file` 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: 387-388, 391, 395-397)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_coverage_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the configuration of coverage settings.

Why Needed: Prevents a bug where coverage settings are not applied to all tests.

Key Assertions:

- config.omit_tests_from_coverage is set to False
- config.include_phase is set to "all"
- asserts that omit_tests_from_coverage is False
- asserts that include_phase is set to "all"
- asserts that 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: 387-388, 391, 395-397)

PASSED tests/test_options_extended.py::TestConfigAnnotations::test_custom_exclude_globs 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `llm_context_exclude_globs` attribute is populated with custom exclude globs.

Why Needed: This test prevents a potential bug where the `llm_context_exclude_globs` attribute is not correctly set for custom exclude globs.

Key Assertions:

- The `*.pyc` glob should be included in the `llm_context_exclude_globs` list.
- The `*.log` glob should be included in the `llm_context_exclude_globs` list.
- The `*.class` glob (not shown) should also be included in the `llm_context_exclude_globs` list if it is a custom exclude glob.
- If no custom exclude globs are provided, the `llm_context_exclude_globs` attribute should still contain the default exclude globs (`*.pyc`, `*.log`), which are not included in this test.
- The `exclude_globs` parameter of the `Config` class should be able to accept a list of custom glob patterns.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_include_globs` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `include_globs` configuration option includes only Python files.

Why Needed: This test prevents a potential bug where the `include_globs` option is not properly configured, potentially leading to incorrect or incomplete model training.

Key Assertions:

- The `*.py` glob matches any file with a `.py` extension.
- The `*.pyi` glob matches any file with an `.pyi` extension.
- The `include_globs` configuration option is applied to the LLM context.
- The `llm_context_include_globs` attribute contains the specified globs.
- The `include_globs` value does not include any files that do not have a `.py` or `.pyi` extension.
- The `include_globs` configuration option is applied to the LLM context with the correct glob patterns.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_invocation_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that `include_pytest_invocation` is set to `False` for the specified configuration.

Why Needed: This test prevents a potential bug where `include_pytest_invocation` is incorrectly set to `True` by an external factor, causing unexpected behavior in tests.

Key Assertions:

- The `include_pytest_invocation` attribute of the `Config` object is set to `False`.
- The `include_pytest_invocation` attribute of the `Config` object does not match its expected value (`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: 107)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_llm_execution_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the LLM execution settings are correctly configured.

Why Needed: This test prevents a potential bug where the maximum tests, concurrency, requests per minute, and timeout seconds are not properly set for the LLM.

Key Assertions:

- The value of llm_max_tests is indeed 50.
- The value of llm_max_concurrency is indeed 8.
- The value of llm_requests_per_minute is indeed 12.
- The maximum tests configured in the test are not exceeded.
- The concurrency limit is respected for each test.
- The requests per minute threshold is respected within the allowed range.
- The timeout seconds are respected within the allowed range.
- A cache directory of the correct path (.cache) 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_llm_param_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the configuration of LLM parameter settings.

Why Needed: Prevents a potential bug where the maximum character limit for LLM parameters is not respected.

Key Assertions:

- config.llm_include_param_values should be set to True
- config.llm_param_value_max_chars should equal 200
- config.llm_include_param_values should be a boolean value
- config.llm_param_value_max_chars should be an integer value
- config.llm_param_value_max_chars should not exceed 200
- config.llm_include_param_values should only include parameter 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: 387-388, 391, 395-397)

PASSED tests/test_options_extended.py::TestConfigAnnotations::test_llm_settings 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the configuration of LLM settings with OLLAMA provider.

Why Needed: This test prevents a bug where the LLM context bytes are not set correctly for the OLLAMA provider.

Key Assertions:

- The value of `config.llm_context_bytes` is equal to 64000 (the expected default value).
- The value of `config.provider` is equal to 'ollama' (the expected default provider).
- The value of `config.model` is equal to 'llama3.2' (the expected default model).

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED tests/test_options_extended.py::TestConfigAnnotations::test_repo_root_path 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `repo_root` attribute is correctly set to `/project` when a `Config` object is created with this configuration.

Why Needed: This test prevents a potential bug where the `repo_root` attribute is not set correctly, potentially leading to incorrect repository root detection.

Key Assertions:

- The `repo_root` attribute of the `Config` object should be equal to `/project`.
- The `repo_root` attribute of the `Config` object should be a `Path` object representing the absolute path `/project`.
- The `repo_root` attribute of the `Config` object should not be `None` or an empty string.
- The `repo_root` attribute of the `Config` object should be a valid directory path.
- The `repo_root` attribute of the `Config` object should not be a relative path.
- The `repo_root` attribute of the `Config` object should not be set to an absolute path that is already in use by another configuration.
- The `repo_root` attribute of the `Config` object should be correctly set when using a `Path` object as the repository 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	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_valid_phase_values` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that all valid include_phase values pass validation.

Why Needed: This test prevents a potential bug where invalid or missing include_phase values cause the validation to fail.

Key Assertions:

- The validate() method of the Config object should not return any errors for include_phase values set to 'run', 'setup', and 'teardown'.
- The validate() method of the Config object should return an empty list for include_phase values set to 'all'.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_exclude_globs` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the default exclude globs for llm context.

Why Needed: Prevents a potential bug where default exclude globs are not correctly set.

Key Assertions:

- The function `defaults` should contain the glob pattern `*.pyc`.
- The function `defaults` should contain the glob pattern `__pycache__/*`.
- The function `defaults` should contain the glob pattern `*secret*`.
- The function `defaults` should contain the glob pattern `*password*`.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_redact_patterns` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test default redact patterns in Config DefaultsMaximal test.

Why Needed: Prevents a potential security vulnerability by ensuring that sensitive information is not exposed when defaulting to minimal configuration.

Key Assertions:

- The `--password` pattern should be found in the default redact patterns.
- The `--token` pattern should be found in the default redact patterns.
- The `--api[_-]?key` pattern should be found in the default redact patterns.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_values` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test default values of the configuration.

Why Needed: To ensure that the default values are correct and do not contain any errors.

Key Assertions:

- The provider should be set to 'none'.
- The llm_context_mode should be set to 'minimal'.
- The llm_context_bytes should be set to 32000.
- The omit_tests_from_coverage flag should be True.
- The include_phase flag 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: 387-388, 391, 395-397)

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 bug where the test fails when using 'ollama' provider due to incorrect implementation of `is_llm_enabled` method.

Key Assertions:

- The `is_llm_enabled` method should return False for the 'none' provider.
- The `is_llm_enabled` method should return True for the 'ollama', 'litellm', and 'gemini' providers.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_aggregate_policy` 1ms 🛡️ 3

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, causing unexpected behavior or errors.

Key Assertions:

- The function `Config` should be instantiated with a valid aggregate policy.
- The error message for an invalid aggregate policy should include the specific policy being used.
- At least one error should be returned when validating an invalid aggregate 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: 387-388, 391, 395-397)`

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_context_mode` 1ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed after 3 retries. Last error: 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/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: 387-388, 391, 395-397)`

PASSED tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_include_phase 1ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed after 3 retries. Last error: 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/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: 387-388, 391, 395-397)

PASSED tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_provider 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test validates an invalid provider.

Why Needed: Prevents a potential bug where the test fails with an error message indicating an invalid provider.

Key Assertions:

- The function `Config` is called with an invalid provider.
- An error message is returned when the provider is invalid.
- The error message contains the string 'Invalid provider'.
- The number of errors returned is correct (1).
- The first error message contains the string 'invalid'.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_numeric_bounds` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_numeric_bounds

Why Needed: Prevents regression where the llm_context_bytes value is set to a non-integer or negative number.

Key Assertions:

- The 'llm_context_bytes' field should be an integer and not exceed 2^31-1.
- The 'llm_max_tests' field should be an integer greater than 0.
- The 'llm_requests_per_minute' field should be an integer greater than 0.
- The 'llm_timeout_seconds' field should be an integer greater than 0.
- All fields should not exceed their maximum allowed values (2^31-1, 10^9, etc.).

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_valid_config` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `validate` method returns an empty list for a valid configuration.

Why Needed: Prevents potential infinite recursion in case of invalid configurations.

Key Assertions:

- The `validate` method should return an empty list when given a valid configuration.
- No exceptions should be raised if the input is valid.
- The method should not throw any errors or warnings for well-formed configurations.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_plugin_integration.py::TestPluginConfigLoading::test_config_defaults` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `load_config` function returns a valid configuration object with default settings.

Why Needed: This test prevents a potential bug where the plugin's configuration is missing or has unexpected defaults if no options are provided.

Key Assertions:

- The `cfg` variable should be an instance of `Config`.
- The `cfg` variable should have some default settings (e.g., `base_dir`, `output_dir`).
- If `pytestconfig` is not a valid pytest configuration, the test will fail with a meaningful error message.
- The `cfg` object's attributes (`base_dir`, `output_dir`) should be set to their default values.
- The `cfg` object's attributes (`log_level`, `verbose`, etc.) should have some default values (e.g., `INFO`, `DEBUG`).
- If the plugin has custom settings, they should override the defaults if provided in `pytestconfig`.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_plugin_integration.py::TestPluginConfigLoading::test_markers_exist_in_config` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Verify that markers in the configuration are accessible.

Why Needed: Prevent a potential bug where markers are not found or are inaccessible in the configuration.

Key Assertions:

- pytestconfig should be an instance of pytest.config.Config
- pytestconfig should have a __dict__ attribute
- markers should be accessible through pytestconfig
- markers should be accessible through pytestconfig.__dict__
- markers should be accessible through pytestconfig.config
- markers should be accessible through pytestconfig._config
- markers should be accessible through pytestconfig._config.__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/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_plugin_integration.py::TestPluginIntegration::test_llm_context_marker` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test the functionality of LLM context marker in a test.

Why Needed: The LLM context marker should prevent errors caused by incorrect usage or configuration of the plugin.

Key Assertions:

- assert True is not raised when calling `test_llm_context_marker`
- the output of `pytest_llm_report/collector.py` does not contain any error messages
- the LLM context marker is properly applied to test cases without causing 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_plugin_integration.py::TestPluginIntegration::test_llm_opt_out_marker` 1ms 🛡️ 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_plugin_integration.py::TestPluginIntegration::test_requirement_marker 1ms 🛡️ 2

AI ASSESSMENT

Scenario: The `requirement_marker` function is being tested to ensure it does not throw any errors when called.

Why Needed: This test prevents a potential bug where the `requirement_marker` function might cause an error if not used correctly.

Key Assertions:

- The `requirement_marker` function should be able to handle its input without throwing any exceptions.
- The `requirement_marker` function should not throw any errors when called with valid inputs.
- Any unexpected behavior or side effects of the `requirement_marker` function should be avoided.
- The `requirement_marker` function should not raise any exceptions when executed successfully.
- The `requirement_marker` function's output should be consistent and predictable in all cases.
- Any potential bugs or regressions caused by changes to the `requirement_marker` function should be identified and fixed.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)

PASSED `tests/test_plugin_integration.py::TestReportGeneration::test_report_writer_integration` 34ms 🛡️ 6

AI ASSESSMENT

Scenario: The test verifies that the report writer correctly generates a full report with summary statistics.

Why Needed: This test prevents regression where the report writer fails to generate a correct report even when there are multiple tests with different outcomes.

Key Assertions:

- Verify that the total number of tests is 2 (1 passed, 1 failed).
- Verify that only 'test_a.py' and 'test_b.py' are included in the report HTML.
- Check if the report JSON file exists and contains a summary with correct statistics (total: 2, passed: 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/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: 387-388, 391, 395-397)
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 `pytest_collectreport` skips when the collect report is disabled.

Why Needed: This test prevents a regression where the collect report is enabled but still causes issues with pytest.

Key Assertions:

- The `pytest_collectreport` function should not be called with an empty stash.
- The `session.config.stash.get` method should return False for `_enabled_key` when it's disabled.
- The `pytest_collectreport` function should not be called multiple times with the same stash key.
- The `session.config.stash.get` method should not call `pytest_collectreport` again after returning False.
- The `pytest_collectreport` function should only be called once per test run when collect report is disabled.
- The `session.config.stash.get` method should return True for `_enabled_key` when it's 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: 387-388, 391, 395-397, 408-409, 415-416)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_enabled` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that collectreport calls collector when enable is True.

Why Needed: To prevent a potential bug where the plugin does not call the collector when enabled.

Key Assertions:

- The `pytest_collectreport` function should be called with the correct mock report instance.
- The `handle_collection_report` method of the mock collector should be called once with the provided mock report.
- The `stash_get` function should return True for the `_enabled_key` and `_collector_key` keys when enabled.
- The `stash_get` function should not return a default value for the `_enabled_key` key when enabled.
- The `handle_collection_report` method of the mock collector should be called only once with the provided mock report.
- The `stash_get` function should return True for the `_collector_key` key when collectreport is called.
- The `stash_get` function should not return a default value for the `_collector_key` key when collectreport is called.
- The `handle_collection_report` method of the mock collector should be called only once with the provided mock 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	12 lines (ranges: 387-388, 391, 395-397, 408-409, 415, 419-421)

PASSED tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_no_session 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Verifies that `pytest_collectreport` does not raise an exception when no session is available.

Why Needed: Prevents a potential bug where the plugin raises an error due to missing a required attribute.

Key Assertions:

- The function `pytest_collectreport` should not be called with a `session` argument that is missing.
- A `Session` object is expected to have a `session` attribute.
- Without a `session` attribute, the plugin will not raise an exception.
- If no session is available, the test should pass without raising any 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/plugin.py	8 lines (ranges: 387-388, 391, 395-397, 408, 412)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_session_none` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Verify that the `pytest_collectreport` function skips when a null session is provided.

Why Needed: Prevent regression in case of missing sessions during Pytest collection.

Key Assertions:

- The `pytest_collectreport` function should not raise an exception when given a `session` attribute set to `None`.
- A null `session` attribute should be ignored by the function.
- The function should skip any collected reports without a valid session.
- No error message or indication of failure should be raised in this case.
- The function's behavior should not depend on the presence of a `pytest_collectreport` instance.
- A null `session` attribute should have no impact on the report collection 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/plugin.py`	`8 lines (ranges: 387-388, 391, 395-397, 408, 412)`

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_llm_enabled_warning` 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that LLM enabled warning is raised when using the Ollama LLMS provider.

Why Needed: This test prevents a potential regression where the LLM report provider 'ollama' might be enabled by default, potentially causing issues with the plugin's functionality.

Key Assertions:

- The `pytest_llm_report` option is set to `None` when using the Ollama LLMS provider.
- The `llm_report_provider` option is set to 'ollama' when using the Ollama LLMS provider.
- The `llm_report_html`, `llm_report_json`, and `llm_report_pdf` options are not set when using the Ollama LLMS provider.
- The `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 are not set when using the Ollama LLMS provider.
- The `llm_max_retries` option is set to `None` when using the Ollama LLMS 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	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, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_validation_errors` 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that validation errors raise UsageError when configuring Pytest.

Why Needed: Prevents configuration errors and ensures correct usage of the plugin.

Key Assertions:

- Mocking `pytest_configure` with invalid config values should raise a `UsageError`.
- The `option.llm_report_html`, `option.llm_report_json`, `option.llm_report_pdf`, `option.llm_evidence_bundle`, `option.llm_dependency_snapshot`, `option.llm_requests_per_minute`, `option.llm_aggregate_dir`, `option.llm_aggregate_policy`, `option.llm_aggregate_run_id`, and `option.llm_aggregate_group_id` options should be set correctly.
- The `llm_report_provider` option should have a valid value.
- Mocking the `getini` method with invalid config values should not affect the behavior of the test.
- The `pytest_configure` function should raise a `UsageError` when called with an invalid 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	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, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_worker_skip` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that configure skips on xdist workers and verifies the correct behavior of addinivalue_line.

Why Needed: This test prevents a potential regression where configure might skip on xdist workers due to an incorrect marker setup.

Key Assertions:

- Mocking pytest_configure with mock_config and checking if addinivalue_line is called.
- Verifying that the workerinput attribute of mock_config is set correctly.
- Checking if the addinivalue_line function is not called before the worker check.
- Asserting that the addinivalue_line function is indeed called after the worker check.

COVERAGE

src/pytest_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, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigureFallback::test_pytest_configure_fallback_load` 3ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that fallback to load_config occurs when Config.load is missing, preventing potential bugs or regressions.

Why Needed: This test prevents a bug where the plugin would attempt to load configuration data without calling Config.load(), potentially causing issues with dependency loading or other related problems.

Key Assertions:

- The `pytest_configure` function should be called with a valid `Config` object.
- The `load_config` method of `Config` should not be called.
- The `validate` method of the mocked `Config` object should return an empty list.
- The `load_config` method of the mocked `Config` object should be called once.
- The `pytest_configure` function should not have been patched with a mock for `options.Config`.
- The `load_config` method of the mocked `Config` object should have been called only once.
- The `validate` method of the mocked `Config` object should have returned 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/plugin.py	29 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 387-388, 391, 395-397)

PASSED tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_all_ini_options 2ms 🛡️ 3

AI ASSESSMENT

Scenario: Test loading all INI options from the plugin configuration.

Why Needed: This test prevents a potential bug where the plugin fails to load configurations with missing or invalid INI values.

Key Assertions:

- The `provider` attribute of the loaded configuration should be set to 'ollama'.
- The `model` attribute of the loaded configuration should be set to 'llama3.2'.
- The `llm_context_mode` attribute of the loaded configuration should be set to 'complete'.
- The `llm_requests_per_minute` attribute of the loaded configuration should be set to 10.
- The `report_html` attribute of the loaded configuration should be set to 'ini.html'.
- The `report_json` attribute of the loaded configuration should be set to '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: 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_cli_overrides_ini` 2ms 🛡️ 3

AI ASSESSMENT

Scenario: Test CLI options override INI options.

Why Needed: This test prevents regression where CLI options override INI options, ensuring that the correct report is generated based on user input.

Key Assertions:

- The `llm_report_html` option overrides the `report_html` value from the INI file.
- The `llm_report_json` option overrides the `report_json` value from the INI file.
- The `llm_report_pdf` option overrides the `report_pdf` value from the INI file.
- The `llm_evidence_bundle` option overrides the `report_evidence_bundle` value from the INI file.
- The `llm_dependency_snapshot` option overrides the `report_dependency_snapshot` value from the INI file.
- The `llm_requests_per_minute` option overrides the `llm_report_requests_per_minute` value from the INI file.
- The `llm_aggregate_dir` option overrides the `aggregate_dir` value from the INI file.
- The `llm_aggregate_policy` option overrides the `aggregate_policy` value from the INI file.
- The `llm_aggregate_run_id` option overrides the `aggregate_run_id` value from the INI file.
- The `llm_aggregate_group_id` option overrides the `aggregate_group_id` value from the INI file.
- The `rootpath` option is set to `/project` 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/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: 387-388, 391, 395-397)

PASSED tests/test_plugin_maximal.py::TestPluginMaximal::test_terminal_summary_disabled 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that terminal summary skips when plugin is disabled.

Why Needed: Prevents a regression where the plugin's terminal summary is not properly handled when it is disabled.

Key Assertions:

- Mock stash.get() was called with _enabled_key and False as arguments, indicating that the plugin should be enabled for this test.
- The stash.get() call did not include any key-value pairs, suggesting that the plugin's terminal summary is being skipped due to its disabled state.

COVERAGE

src/pytest_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, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::test_terminal_summary_worker_skip` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that terminal summary skips on xdist worker when the 'workerid' key is present in the mock configuration.

Why Needed: This test prevents a regression where the plugin does not skip terminal summaries if an xdist worker is configured with a specific ID.

Key Assertions:

- The function `pytest_terminal_summary` should return None for the given mock configuration.
- The 'workerid' key in the mock configuration should be present.
- The value of the 'workerid' key in the mock configuration should be 'gw0'.
- No output or exception should be raised when calling `pytest_terminal_summary` with the given mock configuration and arguments.
- The function `pytest_terminal_summary` should not call any other functions or methods that are not part of the plugin's API.
- The function `pytest_terminal_summary` should not modify any external state or variables.
- No assertion errors should be raised when calling `pytest_terminal_summary` with the given mock configuration and 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/plugin.py	8 lines (ranges: 238-239, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::testload_config` 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Test config loading from pytest objects (CLI + INI) to ensure correct configuration retrieval.

Why Needed: This test prevents a potential bug where the plugin does not retrieve the correct configuration settings due to incorrect handling of INI files.

Key Assertions:

- The `load_config` function should be able to correctly load the specified configuration options from INI files.
- The `getini` method should return None for unknown keys in the INI file, allowing the plugin to handle it appropriately.
- The plugin's rootpath attribute should be set correctly based on the loaded configuration settings.
- The report HTML option should be set to the expected value 'out.html' as per the test setup.
- The `load_config` function should not raise an exception when encountering unknown keys in the INI 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`	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: 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_disabled` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test makereport skips when disabled.

Why Needed: This test prevents a regression where the plugin does not report any issues when makereport is disabled.

Key Assertions:

- The `pytest_runtest_makereport` hookwrapper should be able to send a result and complete even when makereport is disabled.
- The `get_result` method of the mock_outcome object should return None or an empty tuple when called without any arguments.
- The `send` method of the mock_outcome object should not raise an exception when called with no arguments.
- The `pytest_runtest_makereport` hookwrapper should be able to handle the generator and yield a point even when makereport is disabled.
- The `stash.get` method of the mock_item object should return False when called without any arguments.
- The `get_result` method of the mock_outcome object should not raise an exception when called with no arguments.
- The `pytest_runtest_makereport` hookwrapper should be able to handle the generator and yield a point even when makereport is disabled.
- The `pytest_runtest_makereport` hookwrapper should be able to send a result and complete even when makereport 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: 387-388, 391-392, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_enabled` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test makereport calls collector when enabled.

Why Needed: This test prevents a potential bug where the collector is not called when makereport is enabled.

Key Assertions:

- The `mock_collector` is called with the correct `mock_report` instance when `pytest_runtest_makereport` is called.
- The `mock_item.config.stash.get(_enabled_key)` method returns `True` for `_enabled_key` and `mock_collector`.
- The `mock_item.config.stash.get(_collector_key)` method returns `mock_collector` for `_collector_key`.
- The `mock_collector.handle_runtest_logreport()` method is called with the correct `mock_report` instance.
- The `mock_collector.handle_runtest_logreport()` method does not raise an exception when `mock_outcome.get_result()` raises a StopIteration 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: 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_disabled` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that collection_finish is skipped when disabled for Pytest sessions.

Why Needed: This test prevents a regression where the plugin's hooks are not executed correctly when collection_finish is disabled.

Key Assertions:

- mock_session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_collection_finish(mock_session) should be called with _enabled_key as False
- The pytest_collection_finish function should return False for the given key
- The pytest_collection_finish function should not call stash.get again after returning False
- The pytest_collection_finish function should not call pytest_collection_finish again

COVERAGE

`src/pytest_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: 387-388, 391, 395-397, 431-432)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_enabled` 2ms 🛡 2

AI ASSESSMENT

Scenario: Test that collection_finish is called when Pytest collection finish is enabled.

Why Needed: This test prevents a regression 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.
- The mock_collector.handle_collection_finish method should not be called if the key is not _enabled_key or _collector_key.

COVERAGE

src/pytest_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: 387-388, 391, 395-397, 431, 435-437)

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: Prevents a potential bug where the plugin fails to check the enabled status of the session.

Key Assertions:

- mock_session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_sessionstart(mock_session).should.have.been.called
- mock_session.config.stash.get.return_value.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/plugin.py	8 lines (ranges: 387-388, 391, 395-397, 448-449)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_enabled` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that sessionstart initializes collector when enabled and stash supports both get() and [].

Why Needed: Prevents a bug where the collector is not created even though sessionstart is enabled, potentially leading to incorrect data collection.

Key Assertions:

- assert _collector_key in mock_stash
- assert _start_time_key in mock_stash

COVERAGE

`src/pytest_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: 387-388, 391, 395-397, 448, 452, 455, 457-458)

PASSED tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test pytest_addoption adds expected arguments and verifies specific options.

Why Needed: This test prevents a potential bug where pytest_addoption does not add the required 'LLM-enhanced test reports' option to the command line.

Key Assertions:

- Verify that `pytest_addoption` is called with the correct group.
- Verify that the 'LLM-enhanced test reports' option is added to the command line.
- Verify that the 'LLM-coverage-source' option is also added to the command line.
- Verify that any arguments starting with '--llm-report' are included in the list of options.
- Verify that any arguments starting with '--llm-coverage-source' are included in the list of options.

COVERAGE

src/pytest_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	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, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption_ini` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test pytest_addoption adds INI options (lines 13-34) for a plugin with a custom terminal summary.

Why Needed: This test prevents regression when using the pytest_addoption to add INI options to the plugin's configuration.

Key Assertions:

- The 'llm_report_html' option is added to the parser.
- The 'llm_report_json' option is added to the parser.
- The 'llm_report_max_retries' option is added to the parser.
- The 'max_retries' option is present in the ini calls.
- The 'html' and 'json' options are included in the ini calls.
- The 'max_retries' option has a value of 3 or more (>= 2)
- The 'max_retries' option does not have any default value (i.e., it is an optional argument)

COVERAGE

src/pytest_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	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, 387-388, 391, 395-397)

PASSED tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_coverage_calculation 4ms 🛡️ 3

AI ASSESSMENT

Scenario: Test coverage percentage calculation logic for terminal summary.

Why Needed: Prevents regression in coverage reporting of terminal summaries.

Key Assertions:

- The `report_html` option is set to 'out.html' and the `CoverageMapper` is correctly loaded.
- The `report` method is called with a mock configuration object.
- The `Coverage` class reports a percentage of 85.5 coverage.
- The `CoverageMapper` calls `load` before calling `report` on the `Coverage` instance.
- The `ReportWriter` does not raise an exception when writing to the file.
- The mock configuration object has a correct `stash` attribute.
- The mock stash is correctly populated with the provided data.

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	58 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, 309-315, 317-318, 331-332, 337-338, 365-375, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_llm_enabled` 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Test terminal summary with LLM enabled runs annotations.

Why Needed: Prevents regression in terminal summary functionality when LLM is enabled.

Key Assertions:

- Verify that the correct provider is used when LLM is enabled.
- Check if the correct configuration is passed to the terminal reporter.
- Assert that the annotate_tests function is called with the correct config.
- Verify that the get_provider function returns the correct model name.
- Ensure that the mock stash object is correctly populated and mocked.
- Test that the mock_terminalreporter.stats dictionary is not modified unexpectedly.
- Verify that the mock_annotate function is called once with the correct arguments.

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, 331-332, 337-340, 343, 345, 348-350, 357-362, 365-375, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_no_collector` 2ms 🛡️ 3

AI ASSESSMENT

Scenario: Test terminal summary creates collector when no collector is provided.

Why Needed: Prevents a potential bug where the plugin does not create a collector even though it's enabled.

Key Assertions:

- assert stash._enabled_key == True, 'Expected stash to have _enabled_key set to True'.
- assert stash._config_key is None, 'Expected stash to have _config_key set to None'.
- assert mock_config.stash._enabled_key == False, 'Expected stash._enabled_key to be False'.
- assert mock_config.stash._config_key is None, 'Expected stash._config_key to be None'.
- assert mock_terminalreporter.call_args_list[0][1] == 0, 'Expected terminal reporter call with argument 0 to pass'.
- assert mock_writer_cls.return_value.call_args_list[0][1] == 0, 'Expected writer class call with argument 0 to pass'.
- assert mock_mapper_cls.return_value.call_args_list[0][1].get('coverage') is None, 'Expected coverage map to be empty when no collector is provided'.

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, 331, 337-338, 365-375, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_with_aggregation` 2ms 🛡️ 3

AI ASSESSMENT

Scenario: Test terminal summary with aggregation enabled.

Why Needed: This test prevents regression in case the aggregation feature is not properly configured or disabled.

Key Assertions:

- The aggregation key should be set to True.
- The stash should support both get() and [] indexing.
- The aggregation method should return a report.
- The ReportWriter should write JSON and HTML files.
- The aggregation function should not raise an exception when called with invalid 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/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, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummaryErrors::test_terminal_summary_coverage_error` 4ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed after 3 retries. Last error: Failed to parse LLM response as JSON

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, 322-325, 331-332, 337-338, 365-375, 387-388, 391, 395-397)

PASSED `tests/test_prompts.py::TestContextAssembler::test_assemble_balanced_context` 8ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the ContextAssembler to assemble a balanced context for a test file with dependencies.

Why Needed: This test prevents regressions where the ContextAssembler fails to assemble a balanced context due to missing or incorrect dependencies.

Key Assertions:

- The 'utils.py' file is present in the assembled context.
- The 'def util()' function is found in the 'utils.py' file within the assembled context.
- The test file 'test_a.py' has a dependency on 'utils.py'.
- The ContextAssembler correctly assembles a balanced context with all dependencies included.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
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)

PASSED tests/test_prompts.py::TestContextAssembler::test_assemble_complete_context 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Assembling a complete context for the 'test_a.py' test file

Why Needed: Prevents regression when the 'complete' mode is used with a non-existent test file.

Key Assertions:

- The 'test_1' function should be found in the assembled source code.
- The 'test_1' function should have been executed as part of the context assembly process.
- The 'test_a.py::test_1' nodeid should match the expected location within the assembled 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: 387-388, 391, 395-397)
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)

PASSED tests/test_prompts.py::TestContextAssembler::test_assemble_minimal_context 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Verifies that the ContextAssembler can assemble a minimal context for a test file with a single test function.

Why Needed: This test prevents regression when using the 'minimal' llm_context_mode, as it ensures that only necessary code is assembled into the context.

Key Assertions:

- The source of the assembly contains the expected test function `test_1`.
- The context is empty, indicating no additional code was assembled into the context.
- The assembler correctly identifies the test file `test_a.py` as part of the 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: 387-388, 391, 395-397)
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)

PASSED `tests/test_prompts.py::TestContextAssembler::test_balanced_context_limits` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Ensures the ContextAssembler can assemble a test with balanced context limits.

Why Needed: This test prevents a potential bug where the assembler fails to truncate long content exceeding 20 bytes.

Key Assertions:

- The 'f1.py' file in the test result should contain the original long content.
- The 'f1.py' file in the test result should be truncated after 40 characters (20 bytes + truncation message).
- The length of the 'f1.py' file in the test result should not exceed 40 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: 387-388, 391, 395-397)
`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)

PASSED `tests/test_prompts.py::TestContextAssembler::test_get_test_source_edge_cases` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the ContextAssembler's get_test_source method with edge cases.

Why Needed: This test prevents a potential bug where the assembler incorrectly handles non-existent files or nested test names with parameters.

Key Assertions:

- The function returns an empty string for a non-existent file.
- The function correctly identifies the 'test' keyword in the source code of a nested test name with parameters.
- The function does not return any value when the input is invalid (e.g., no test name or parameter 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	26 lines (ranges: 33, 78-79, 82-84, 86-87, 92, 94-95, 98-101, 103-112, 116)

PASSED `tests/test_prompts.py::TestContextAssembler::test_should_exclude` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the ContextAssembler should exclude certain files from the LLM context.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly excludes important files like `*.pyc` or sensitive data stored in `secret/*`.

Key Assertions:

- assert assembler._should_exclude('test.pyc') is True
- assert assembler._should_exclude('secret/key.txt') is True
- assert 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: 387-388, 391, 395-397)
`src/pytest_llm_report/prompts.py`	5 lines (ranges: 33, 191-194)

PASSED tests/test_ranges.py::TestCompressRanges::test_consecutive_lines 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The `compress_ranges` function is tested to ensure consecutive lines are compressed correctly.

Why Needed: This test prevents a potential bug where consecutive lines of text without spaces or tabs are incorrectly compressed into a single range.

Key Assertions:

- The input list contains at least two elements.
- There are no empty strings in the input list.
- All elements in the input list are integers.
- - The first element is less than or equal to the second element.
- - All elements are within a valid range (inclusive).
- If there's only one element, it should be compressed into a single range ('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: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

PASSED `tests/test_ranges.py::TestCompressRanges::test_duplicates` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the function correctly handles duplicate values in the input range.

Why Needed: This test prevents a potential bug where the function incorrectly identifies unique ranges when there are duplicates.

Key Assertions:

- The function should return '1-3' for the input range [1, 2, 2, 3, 3, 3].
- The function should not return any duplicate ranges (e.g., [1, 2] or [2, 3]).
- The function should handle ranges with an odd number of elements correctly.
- The function should ignore non-integer values in the input range.
- The function should raise a ValueError for invalid input types (e.g., None or 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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)`

PASSED `tests/test_ranges.py::TestCompressRanges::test_empty_list` 1ms 🛡️ 3

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 would cause the function to return an incorrect result or raise an exception.

Key Assertions:

- The function should return an empty string for an empty input list.
- The function should not throw any exceptions when given an empty list as input.
- The function should preserve the original order of elements in the input list.
- The function should handle lists with a single element correctly.
- The function should ignore non-compressible ranges (e.g., [1, 2] and [3, 4])
- The function should not compress ranges that are already empty (e.g., [] and [])
- The function should preserve the original order of elements when multiple non-compressible ranges are 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/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`2 lines (ranges: 29-30)`

PASSED `tests/test_ranges.py::TestCompressRanges::test_mixed_ranges` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the function with a mixed range of numbers (single and multi-range values)

Why Needed: Prevents regression in case of mixed ranges where single values are used as start or end points.

Key Assertions:

- The output should be '1-3, 5, 10-12, 15' as per the expected result.
- The function should correctly handle cases where a range is used as a single value (e.g., [1, 2, 3])
- The function should not raise any errors when given invalid input (e.g., negative numbers or non-numeric values).
- The function should preserve the original order of elements in the input list.
- The function should correctly handle ranges with a single element (e.g., [1, 5])
- The function should not incorrectly group adjacent ranges together (e.g., '1-3' and '5-10').

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/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)`

PASSED `tests/test_ranges.py::TestCompressRanges::test_non_consecutive_lines` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that non-consecutive lines are correctly compressed to a single comma-separated value.

Why Needed: This test prevents regression where consecutive lines are not properly compressed.

Key Assertions:

- Ensure that the input list is correctly split into ranges without any consecutive elements.
- Verify that each range contains all its elements, without gaps.
- Check if the resulting string has no commas between the numbers.
- Confirm that the order of the numbers in each range remains unchanged.
- Test for edge cases where the input list only contains one element or multiple elements with no gaps.
- Ensure the function handles lists with duplicate values correctly.
- Verify the output is as expected even when the input list has a large number of consecutive 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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/ranges.py`	14 lines (ranges: 29, 33, 35-37, 39-40, 45-47, 50, 52, 65-66)

PASSED `tests/test_ranges.py::TestCompressRanges::test_single_line` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: tests/test_ranges.py::TestCompressRanges::test_single_line

Why Needed: This test prevents regression when the input list contains a single element.

Key Assertions:

- The function compress_ranges() should return the expected string representation for a single-element list.
- The function compress_ranges() should not use range notation to represent the list.
- The function compress_ranges() should raise an error if the input is empty.
- The function compress_ranges() should handle lists with only one element correctly.
- The function compress_ranges() should return the correct string representation for a single-element list.
- The function compress_ranges() should not modify the original list.
- The function compress_ranges() should raise an error if the input is not a list.
- The function compress_ranges() should handle lists with multiple elements 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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`10 lines (ranges: 29, 33, 35-37, 39, 50, 52, 65-66)`

PASSED `tests/test_ranges.py::TestCompressRanges::test_two_consecutive` 1ms 🛡️ 3

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 number.

Key Assertions:

- The input list should contain exactly one element.
- The output string should be in the format 'a-b'.
- The range notation should start from the first element and end at the second element.
- The range notation should include both elements.
- The range notation should not include any separators (e.g., commas).
- The input list should contain only one number.
- The output string should be in the correct order (first element before 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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)`

PASSED `tests/test_ranges.py::TestCompressRanges::test_unsorted_input` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the `compress_ranges` function handles unsorted input correctly.

Why Needed: The test prevents a potential bug where the function would incorrectly group ranges in an unsorted array.

Key Assertions:

- The function should return '1-3, 5' as expected when given the input `[5, 1, 3, 2]`.
- The function should correctly group the range `[5, 3]` into `['1', '3']`.
- The function should handle duplicate ranges correctly by not producing any additional groups.
- The function should preserve the original order of elements within each range.
- The function should return an empty string for input that is already sorted.
- The function should raise a ValueError when given invalid input (e.g. non-integer values).
- The function should handle edge cases where the input array has only 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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)`

PASSED `tests/test_ranges.py::TestExpandRanges::test_empty_string` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that an empty string produces an empty list when expanded by the `expand_ranges` function.

Why Needed: This test prevents a potential bug where an empty string is not correctly handled by the `expand_ranges` function, potentially leading to incorrect results or errors.

Key Assertions:

- The `expand_ranges` function should return an empty list when given an empty input string.
- The `expand_ranges` function should handle empty strings without raising any exceptions or producing unexpected results.
- The `expand_ranges` function should produce a correct output for an empty string, such as 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/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`2 lines (ranges: 81-82)`

PASSED `tests/test_ranges.py::TestExpandRanges::test_mixed` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `expand_ranges` function correctly handles mixed ranges and singles.

Why Needed: This test prevents a potential bug where the function incorrectly expands single values into multiple ranges.

Key Assertions:

- The input string should be parsed as a comma-separated list of range strings or integers.
- Each range string should be in the format 'start-end' (e.g., '1-3'),
- Single values should not be expanded into multiple ranges.
- Range start and end values should be consecutive integers.
- Invalid range formats (e.g., 'a-b', 'abc') should raise a ValueError.
- The function should return the correct result for edge cases (e.g., empty input, single value)
- The function should handle ranges with negative numbers 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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`11 lines (ranges: 81, 84-91, 93, 95)`

PASSED `tests/test_ranges.py::TestExpandRanges::test_range` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The 'expand_ranges' function is expected to correctly handle the range '1-3' and return a list of numbers in that range.

Why Needed: This test prevents a potential bug where the function might not expand the range correctly, potentially leading to incorrect results or errors.

Key Assertions:

- The function should return a list containing the numbers from 1 to 3 (inclusive).
- The function should handle negative values in the range. For example, '-2-4' should also be expanded to [-2, -1, 0, 1, 2, 3].
- The function should correctly handle ranges with multiple parts, such as '1-5' or '10-15'.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`10 lines (ranges: 81, 84-91, 95)`

PASSED `tests/test_ranges.py::TestExpandRanges::test_roundtrip` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that `compress_ranges` and `expand_ranges` can be used interchangeably to get the original list back.

Why Needed: This test prevents a potential bug where compressing or expanding ranges would alter the original data in unintended ways.

Key Assertions:

- original == expanded
- compressed == original
- expanded == compressed

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)
`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)

PASSED `tests/test_ranges.py::TestExpandRanges::test_single_number` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The 'expand_ranges' function is expected to handle a single input ('5') and return a list with one element.

Why Needed: This test prevents a potential bug where the function does not correctly handle single numbers, potentially leading to incorrect results or errors.

Key Assertions:

- assert expand_ranges('5') == [5]
- assert isinstance(expand_ranges('5'), list)
- assert len(expand_ranges('5')) == 1
- assert expand_ranges('-5') == [-5]
- assert expand_ranges('abc') == []
- assert expand_ranges('5.0') == [5.0]

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/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`7 lines (ranges: 81, 84-87, 93, 95)`

PASSED tests/test_render.py::TestFormatDuration::test_milliseconds 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the `format_duration` function correctly formats durations for milliseconds below 1 second.

Why Needed: This test prevents a regression where the function does not handle durations less than 1 second correctly.

Key Assertions:

- The function should return '500ms' when given an input of 0.5 seconds.
- The function should return '1ms' when given an input of 0.001 seconds.
- The function should return '0ms' when given an input 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: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	2 lines (ranges: 65, 67)

PASSED `tests/test_render.py::TestFormatDuration::test_seconds` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: tests/test_render.py::TestFormatDuration::test_seconds

Why Needed: This test prevents a potential bug where the function does not correctly format durations for values greater than or equal to 1 second.

Key Assertions:

- The function `format_duration(x)` should return the string 'x.s' when x is an integer and >= 1.
- The function `format_duration(x)` should return the string 'x.00s' when x is a float and >= 1.
- The function `format_duration(x)` should correctly handle values greater than or equal to 1 second by returning the correct format string 'x.s'.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)
`src/pytest_llm_report/render.py`	2 lines (ranges: 65-66)

PASSED `tests/test_render.py::TestOutcomeToCssClass::test_all_outcomes` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test Outcome Mapping to CSS Classes

Why Needed: Prevents regression where outcomes map to different CSS classes.

Key Assertions:

- The `outcome_to_css_class` function should return the correct CSS class for each outcome.
- The `outcome_to_css_class` function should handle all possible outcomes correctly.
- The `outcome_to_css_class` function should not throw any exceptions when given an invalid outcome.
- The `outcome_to_css_class` function should preserve the original outcome value.
- The `outcome_to_css_class` function should map 'passed' to 'outcome-passed', 'failed' to 'outcome-failed', and so on.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	8 lines (ranges: 79-85, 87)

PASSED `tests/test_render.py::TestOutcomeToCssClass::test_unknown_outcome` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the 'outcome_to_css_class' function with an unknown outcome.

Why Needed: Prevents a potential bug where the function returns incorrect CSS classes for unknown outcomes.

Key Assertions:

- The function should return 'outcome-unknown' when given an unknown outcome.
- The function should not return any other class (e.g. 'outcome-foo') when given an unknown outcome.
- The function should handle cases where the outcome is not a string (e.g. a number or None) correctly.
- The function should raise an error if given an invalid outcome (e.g. a non-string value).
- The function should maintain its original behavior for known outcomes (e.g. 'outcome-foo').
- The function should be able to handle cases where the unknown outcome is not in the expected list of valid outcomes.
- The function should be able to handle cases where the unknown outcome is a string that is not recognized by the function (e.g. an invalid CSS class 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/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/render.py`	`8 lines (ranges: 79-85, 87)`

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_basic_report` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test renders basic report with fallback HTML.

Why Needed: This test prevents a rendering issue where the full HTML document is not rendered due to plugin or repository version issues.

Key Assertions:

- The presence of '' in the rendered HTML.
- The text 'Test Report' should be present in the rendered HTML.
- The nodeid 'test::passed' should be found in the rendered HTML.
- The string 'PASSED' should be present in the rendered HTML.
- The string 'FAILED' should be present in the rendered HTML.
- The plugin version should be displayed as '**Plugin:** v0.1.0'.
- The repository version should be displayed as '**Repo:** v1.2.3'.

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: 387-388, 391, 395-397)
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)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_coverage` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the test renders a fallback HTML with coverage information.

Why Needed: This test prevents regression by ensuring that the rendering of `src/foo.py` includes coverage information.

Key Assertions:

- The report root contains a 'Summary' object with a total count of 1 and a passed count of 1.
- The report root contains a 'tests' list with one item, which is a 'TestCaseResult' object.
- The 'CoverageEntry' object has the following properties: 'file_path'='src/foo.py', 'line_ranges'='1-5', and 'line_count'=5.
- The rendered HTML includes the specified file path ('src/foo.py') with 5 lines of code.
- The rendered HTML contains a summary of coverage information, including the total count (1) and passed count (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: 387-388, 391, 395-397)
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)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_llm_annotation` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the report includes LLM annotations for LLMs.

Why Needed: This test prevents a regression where LLM annotations are not included in reports.

Key Assertions:

- The report contains 'Tests login flow' as part of its content.
- The report contains 'Prevents auth bypass' as part of its content.
- The LLM annotation is present and correctly formatted within 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/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`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)`

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_source_coverage` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test renders source coverage for fallback HTML.

Why Needed: Prevents regression where the test fails due to missing source coverage information.

Key Assertions:

- The 'Source Coverage' section is present in the rendered HTML.
- The file path of the source code is included in the 'src/foo.py' tag.
- The overall coverage percentage is displayed as '80.0%', indicating that at least 80% of statements were covered.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)
`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)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_xpass_summary` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the rendered HTML includes both "XFailed" and "XPassed" summaries.

Why Needed: This test prevents a regression where the summary is missing or incorrectly formatted.

Key Assertions:

- The string 'XFailed' should be present in the rendered HTML.
- The string 'XPassed' should also be present in the rendered HTML.
- Both "XFailed" and "XPassed" should appear in the rendered HTML 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/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`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)

PASSED `tests/test_report_writer.py::TestComputeSha256::test_different_content` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'different_content' verifies that the output of `compute_sha256` function is different for two different inputs.

Why Needed: This test prevents a potential bug where the same input could produce the same output, leading to inconsistent reports.

Key Assertions:

- The expected hash values are different.
- The computed hash value does not match the expected one.
- The function `compute_sha256` is correctly calculating the SHA-256 hash of each input.
- The test case is checking for a specific type of mismatch (different content),
- The test case is verifying that the output of the function is different from the input.
- The computed hash value should be unique and not match any other hash value generated by the 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: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

PASSED tests/test_report_writer.py::TestComputeSha256::test_empty_bytes 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that an empty bytes object produces consistent hash and the correct length.

Why Needed: Prevents a bug where different input bytes produce different hashes, potentially leading to incorrect reporting or analysis.

Key Assertions:

- The output of `compute_sha256(b'')` should be equal to `compute_sha256(b'')`.
- The length of the hash produced by `compute_sha256(b'')` should be 64 characters (the expected SHA256 hex length).

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

PASSED `tests/test_report_writer.py::TestReportWriter::test_build_run_meta` 5ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that the build_run_meta method returns the correct metadata for a test run.

Why Needed: This test prevents regression where the report writer does not include version info in the run metadata.

Key Assertions:

- The duration of the test run should be 60 seconds.
- The pytest version should be present in the metadata.
- The plugin version should be '0.1.0'.
- The Python version should also be present in the metadata.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
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)

PASSED `tests/test_report_writer.py::TestReportWriter::test_build_summary_all_outcomes` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test verifies that the `build_summary` method correctly counts all outcome types in a test case.

Why Needed: This test prevents a regression where the summary might incorrectly count some outcomes as 'passed' or 'failed'.

Key Assertions:

- The total number of outcomes should be equal to the number of tests.
- The number of passed outcomes should be 1 (all tests passed).
- The number of failed outcomes should be 1 (all tests failed).
- The number of skipped outcomes should be 1 (one test was skipped).
- The number of xfailed and xpassed outcomes should be 1 each.
- The number of error outcome(s) should be 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: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	19 lines (ranges: 156-158, 312, 314-315, 317-328, 330)

PASSED tests/test_report_writer.py::TestReportWriter::test_build_summary_counts 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that the `build_summary` method correctly counts outcomes in a test case.

Why Needed: This test prevents regression where the total count of passed, failed and skipped tests is not accurate.

Key Assertions:

- The total number of tests should be equal to the sum of passed, failed and skipped tests.
- The number of passed tests should be equal to the number of tests with outcome 'passed'.
- The number of failed tests should be equal to the number of tests with outcome 'failed'.
- The number of skipped tests should be equal to the number of tests with outcome 'skipped'.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	13 lines (ranges: 156-158, 312, 314-315, 317-322, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_create_writer` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that the `ReportWriter` initializes correctly with a given configuration.

Why Needed: This test prevents a potential bug where the `ReportWriter` does not properly initialize with a valid configuration.

Key Assertions:

- The `config` attribute of the `writer` object is set to the provided `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: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	3 lines (ranges: 156-158)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_assembles_tests` 5ms 🛡️ 4

AI ASSESSMENT

Scenario: Test writes a report that includes all tests, but does not handle the case where output paths are provided.

Why Needed: This test prevents a regression where the ReportWriter does not write reports for tests with no output paths.

Key Assertions:

- The length of the report.tests list should be equal to 2.
- The total number of tests in the summary should be equal to 2.
- Each test in the report.tests list should have a nodeid that matches one of the TestCaseResult.nodeids provided.
- Each test result outcome should match either 'passed' or 'failed'.
- The summary.total property should contain an integer value representing the total number of tests.
- The config.output_paths property should be None, indicating no output paths are being 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
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)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_includes_coverage_percent` 6ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class writes a report with an included total coverage percentage.

Why Needed: This test prevents a regression where the coverage percentage is not accurately reflected in the report.

Key Assertions:

- The `coverage_total_percent` attribute of the `report.summary` object should match the provided `coverage_percent` value.
- The `writer.write_report()` method returns an instance of `ReportWriter` with a correctly set `coverage_total_percent` attribute.
- The `report.summary.coverage_total_percent` property is updated to reflect the actual coverage percentage included in 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: 387-388, 391, 395-397)
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)

PASSED tests/test_report_writer.py::TestReportWriter::test_write_report_includes_source_coverage 5ms 🛡 4

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_includes_source_coverage verifies that the test writes a report with source coverage information.

Why Needed: This test prevents regression by ensuring that source coverage is included in reports, which helps maintain data integrity and accuracy.

Key Assertions:

- The length of the `source_coverage` list should be 1.
- The first element of the `source_coverage` list should have a `file_path` attribute equal to 'src/foo.py'.
- All elements in the `source_coverage` list should have a `covered_ranges` attribute that contains at least one range (e.g. '1-4, 6-7').
- The length of each element in the `source_coverage` list should be 3.
- Each element in the `source_coverage` list should have a `missed` attribute equal to 0 or less.
- The value of the `coverage_percent` attribute should be between 0 and 100 (inclusive).
- All elements in the `source_coverage` list should have a `covered` attribute that is greater than 0.
- Each element in the `source_coverage` list should have a `missed_ranges` attribute that contains at least one range (e.g. '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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
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)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_merges_coverage` 5ms 🛡️ 4

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_merges_coverage verifies that the report writer merges coverage into tests.

Why Needed: This test prevents regression where the coverage is not merged into tests, potentially leading to inaccurate reporting of test coverage.

Key Assertions:

- The report should contain a single coverage entry for the specified test.
- The file path of the coverage entry matches the expected file path.
- All lines in the coverage entry have the correct range and 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/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`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)

PASSED tests/test_report_writer.py::TestReportWriterWithFiles::test_atomic_write_fallback 7ms 🛡️ 5

AI ASSESSMENT

Scenario: Test that the ReportWriterWithFiles class falls back to direct write if atomic write fails and reports warnings.

Why Needed: This test prevents a regression where the atomic write operation fails, causing the report writer to fall back to direct writing and reporting warnings.

Key Assertions:

- The file "report.json" should exist at the specified path.
- Any warning messages from the ReportWriterWithFiles class should have code 'W203'.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
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)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_creates_directory_if_missing` 7ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` creates an output directory if it does not exist.

Why Needed: This test prevents a potential bug where the report writer fails to create the output directory when it is missing.

Key Assertions:

- The output directory should be created with the correct name (`subdir/report.json`).
- The `exists()` method of the `tmp_path` object should return True for the expected output directory.
- The `ReportWriter` should correctly create the output directory even if it does not exist.
- The `write_report()` method should write to the expected output file.
- The `json_path` variable should be a valid path to an existing JSON file in the temporary directory.
- The `config` object should have a `report_json` attribute that points to the correct JSON file.
- The `writer.write_report()` method should call the `write()` method of the `Config` class 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/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: 387-388, 391, 395-397)
`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: Test that a directory creation failure results in the capture of a warning.

Why Needed: To prevent unexpected behavior when creating directories with insufficient permissions.

Key Assertions:

- The `writer.warnings` list should contain at least one warning with code 'W201'.
- The `writer.warnings` list should not be empty.
- Any warnings in the `writer.warnings` list should have a non-zero code value ('W201').

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	12 lines (ranges: 156-158, 470-473, 480-484)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_git_info_failure` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `get_git_info` function handles Git command failures by returning `None` for both `sha` and `dirty` variables.

Why Needed: This test prevents a regression where the `get_git_info` function fails to return expected values when encountering a Git command failure.

Key Assertions:

- The `sha` variable is set to `None` after calling `get_git_info()`
- The `dirty` variable is set to `None` after calling `get_git_info()`
- The function does not raise an exception when encountering a Git command failure (as intended)
- The function returns the correct values for `sha` and `dirty` even if the Git command fails (as expected)
- The test does not fail due to a Git command failure (as intended)

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)
`src/pytest_llm_report/report_writer.py`	9 lines (ranges: 67-73, 85-86)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_html_creates_file` 32ms 🛡️ 5

AI ASSESSMENT

Scenario: Test `test_write_html_creates_file` verifies that the report writer creates an HTML file and includes expected content.

Why Needed: This test prevents a regression where the report writer fails to create an HTML file or does not include expected content in the generated report.

Key Assertions:

- The file `report.html` should exist at the specified path.
- The file `report.html` should contain the expected content as per the test cases.
- All test cases (test1 and test2) should be found in the HTML file.
- The report writer should include 'PASSED', 'FAILED', 'Skipped', 'XFailed', and 'XPassed' keywords in the HTML file.
- The report writer should also include 'Errors' keyword 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: 387-388, 391, 395-397)
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)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_html_includes_xfail_summary` 34ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the report includes xfail outcomes in the HTML summary.

Why Needed: This test prevents regression by ensuring that xfail outcomes are included in the report.

Key Assertions:

- Asserts that 'XFAILED' and 'XFailed' are present in the HTML string.
- Asserts that 'XPASSED' and 'XPassed' are also present in the HTML string.
- Verifies that the HTML includes a summary of xfail outcomes.
- Checks if the report is written with the correct configuration.
- Ensures that the test output matches the expected format.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
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)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_json_creates_file` 6ms 🛡️ 5

AI ASSESSMENT

Scenario: Tests the `write_json` method of ReportWriter with a test case that creates a JSON file.

Why Needed: This test prevents regression where the report writer does not create a JSON file.

Key Assertions:

- The function should create a new JSON file at the specified path.
- The file should be tracked as an artifact in the report.
- The number of artifacts 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: 387-388, 391, 395-397)
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)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_pdf_creates_file` 37ms 🛡️ 5

AI ASSESSMENT

Scenario: Test verifies that the `write_pdf` method creates a PDF file when Playwright is available.

Why Needed: This test prevents regression where the `playwright.sync_api` module import fails and the `write_pdf` method does not create a file.

Key Assertions:

- The `write_pdf` function should write to the specified path.
- The file should be created in the same directory as the report.
- Any artifacts generated by the test should have the correct paths.
- The `exists()` method should return True for the PDF file.
- The `artifact.path == str(pdf_path)` assertion should pass for each artifact.
- The `playwright.sync_api` module import should be mocked 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: 387-388, 391, 395-397)
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)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_pdf_missing_playwright_warns` 6ms 🛡️ 4

AI ASSESSMENT

Scenario: Test 'Should warn when Playwright is missing for PDF output' verifies that the test reports a warning when Playwright is not installed for PDF output.

Why Needed: This test prevents a bug where the report writer does not correctly handle cases where Playwright is missing for PDF output, potentially leading to unexpected behavior or errors in the report.

Key Assertions:

- The file 'report.pdf' should exist.
- At least one warning code (W204_PDF_PLAYWRIGHT_MISSING) should be present in the warnings 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
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)

PASSED `tests/test_report_writer_coverage_v2.py::test_report_writer_ensure_dir_creation` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test ensures directory creation of report writer output.

Why Needed: Prevents a potential issue where the report writer creates an empty or non-existent directory.

Key Assertions:

- The `tmp_dir` exists after the test.
- Any warnings from the report writer have a code of 'W202'.
- The `tmp_dir` is not empty after 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: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	11 lines (ranges: 156-158, 470-477)

PASSED tests/test_report_writer_coverage_v2.py::test_report_writer_metadata_skips 10ms 🛡️ 5

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 can lead to incorrect or incomplete 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: 387-388, 391, 395-397)
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)

PASSED `tests/test_schemas.py::TestAnnotationSchema::test_from_dict_full` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that `AnnotationSchema.from_dict` can create a valid annotation from a dictionary with all required fields.

Why Needed: Prevents regression in cases where the input data does not contain all required fields, potentially causing authentication issues.

Key Assertions:

- The schema should have the correct scenario and why needed values.
- All required key assertions should be present and match the expected values.
- The confidence value should match the provided confidence.
- The schema's `scenario` field should match the input dictionary's `scenario` value.
- The schema's `why_needed` field should match the input dictionary's `why_needed` value.
- All required key assertions in the schema's `key_assertions` list should be present and match the expected values.
- Each required key assertion in the schema's `key_assertions` list should have a matching value in the input 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/llm/schemas.py`	5 lines (ranges: 77-81)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_schemas.py::TestAnnotationSchema::test_to_dict_full` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: test_to_dict_full verifies that the annotation schema can be successfully converted to a dictionary with all required fields.

Why Needed: This test prevents regression in the AnnotationSchema class, ensuring it correctly handles the conversion of annotations to dictionaries.

Key Assertions:

- assert data['scenario'] == 'Verify login',
- assert data['why_needed'] == 'Catch auth bugs',
- assert data['key_assertions'] == ['assert 200', 'assert token'],
- assert 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: 387-388, 391, 395-397)`

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_report_created` 87ms 🛡️ 7

AI ASSESSMENT

Scenario: The test verifies that an HTML report is created when the `--llm-report` flag is used.

Why Needed: This prevents a regression where the report generation might not work as expected due to missing or corrupted files.

Key Assertions:

- The file path of the generated report should be present in the current working directory.
- The content of the report should contain the string ''.
- The test function 'test_simple' should be found within 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, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)

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_smoke_pytester.py::TestBasicReportGeneration::test_html_summary_counts_all_statuses 126ms 🛡️ 7

AI ASSESSMENT

Scenario: tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_summary_counts_all_statuses

Why Needed: This test prevents regression by ensuring that the HTML summary counts include all statuses.

Key Assertions:

- assert True is included in the 'Passed' label
- assert False is included in the 'Failed' label
- assert True is included in the 'Skipped' label
- assert True is included in the 'XFailed' label
- assert True is included in the 'Errors' and 'Error' labels

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, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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)

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_json_report_created` 75ms 🛡️ 7

AI ASSESSMENT

Scenario: The JSON report is created and its existence and contents are verified.

Why Needed: This test prevents a potential bug where the Pytest report generation process fails to create a JSON report, potentially leading to incorrect or incomplete reporting.

Key Assertions:

- A JSON file named 'report.json' should be created in the report directory.
- The contents of the 'report.json' file should contain the expected schema version and summary statistics.
- The total number of tests passed should match the actual count, with no failed or skipped tests.
- At least one test should have been marked as 'passed', and at most one test should be marked as '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, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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)

PASSED tests/test_smoke_pytester.py::TestBasicReportGeneration::test_llm_annotations_in_report 83ms 🛡️ 13

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report for a provider enabled.

Why Needed: Prevents regressions by ensuring LLM annotations are present in reports.

Key Assertions:

- The scenario 'Checks the happy path' is included in the report.
- The reason 'Prevents regressions' is included in the report.
- The key assertions 'asserts True' are included in the report.
- The provider is enabled (litellm is used).
- The LLM report model is set to gpt-4o-mini.
- The LLM report JSON file is created at the specified path.

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, 331, 337-340, 343, 345, 348-352, 355, 357-362, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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)

PASSED tests/test_smoke_pytester.py::TestBasicReportGeneration::test_llm_error_is_reported 6.09s 🛡️ 12

AI ASSESSMENT

Scenario: Test 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 'LLM error' is present in the report output.
- The test checks for the correct spelling of 'boom' in the report content.
- The test ensures that both 'LLM error' and 'boom' are found in the report content.
- The test verifies that the LLM errors are surfaced correctly in HTML format.
- The test asserts that the LLM errors are reported with proper formatting.

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, 331, 337-340, 343, 345, 348-353, 357-362, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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)

PASSED tests/test_smoke_pytester.py::TestMarkers::test_llm_opt_out_marker 65ms 🛡️ 7

AI ASSESSMENT

Scenario: Test the LLM opt-out marker functionality.

Why Needed: Prevents regression in LLM opt-out marker detection.

Key Assertions:

- The test verifies that the LLM opt-out marker is correctly recorded.
- The test checks if the LLM opt-out marker is marked as True for all tests.
- The test asserts that only one test is marked with the LLM opt-out marker.
- The test reads and parses the report file to verify the expected data.
- The test verifies that the LLM opt-out marker is correctly recorded in the report file.
- The test checks if the LLM opt-out marker is not marked as False for any tests.
- The test ensures that the LLM opt-out marker is not missed by the pytester.
- The test verifies that the LLM opt-out marker is correctly recorded in the report file even when the --llm-report-json flag is not used.

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, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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)

PASSED tests/test_smoke_pytester.py::TestMarkers::test_requirement_marker 65ms 🛡️ 7

AI ASSESSMENT

Scenario: Test the requirement marker to ensure it records the correct requirements.

Why Needed: This test prevents a regression where the requirement marker is not recorded correctly, potentially leading to missed tests or incorrect test results.

Key Assertions:

- The `pytest.mark.requirement` decorator is applied to the `test_with_req` function with two required requirements.
- The report generated by `pytester.runpytest` includes a JSON file that contains the list of tested functions and their corresponding requirements.
- The test asserts that there is only one test function in the report, which should contain both `REQ-001` and `REQ-002` as required.
- The test asserts that both `REQ-001` and `REQ-002` are present in the list of requirements for the first tested function.
- The test verifies that `REQ-001` is included in the requirements of the `test_with_req` function.

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, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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)

PASSED tests/test_smoke_pytester.py::TestOutcomes::test_multiple_xfail_outcomes 68ms 🛡️ 7

AI ASSESSMENT

Scenario: The test verifies that multiple xfailed tests are recorded in the report.

Why Needed: This test prevents regression by ensuring that all xfailed tests are properly reported and counted.

Key Assertions:

- The number of xfailed tests is correctly reported as 2.
- All xfailed tests are included in the report.
- Each xfailed test has an outcome of 'xfailed'.
- No other outcomes are recorded in the report for these tests.
- The report includes a summary section with the correct count of xfailed tests.

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, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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)

PASSED tests/test_smoke_pytester.py::TestOutcomes::test_skip_outcome 59ms 🛡 7

AI ASSESSMENT

Scenario: Test that skipping tests prevents incorrect reporting of skipped outcomes.

Why Needed: This test verifies that skipping tests is recorded in the report and does not incorrectly count them as passed.

Key Assertions:

- The 'skipped' key in the report should contain a value of 1 if there are any skipped tests.
- The 'summary' section of the report should have a 'skipped' category with a count equal to the number of skipped tests.
- If no tests were skipped, the 'summary' section of the report should be empty.
- If all tests passed, the 'summary' section of the report should also be empty.
- The test skip marker should indicate that the test was skipped.
- The test skip marker should not be present if there are no skipped tests.
- The test skip marker should not be present in the 'skipped' category if all tests passed.

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, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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)

PASSED `tests/test_smoke_pytester.py::TestOutcomes::test_xfail_outcome` 63ms 🛡️ 7

AI ASSESSMENT

Scenario: Verifies that the 'xfail' marker correctly records X-failed tests in the report.

Why Needed: This test prevents regression where a test is marked as xfail but does not actually fail, potentially leading to incorrect reporting of failed tests.

Key Assertions:

- The 'summary' key in the JSON report should contain the correct number of X-failed tests (1 in this case).
- The 'xfailed' value under the 'summary' key should be equal to 1.
- The 'test_results' key under the 'summary' key should include a list of test names that failed, which in this case is just 'test_xfail'.

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, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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)

PASSED `tests/test_smoke_pytester.py::TestParametrization::test_parametrized_tests` 65ms 🛡️ 7

AI ASSESSMENT

Scenario: Test parameterized tests are recorded separately.

Why Needed: This test prevents regression in parametrized testing.

Key Assertions:

- The function `test_param` is called with the correct argument value (`x`) for each iteration.
- The assertion `assert x > 0` passes for all values of `x` (1, 2, and 3).
- The test case has a total count of 3 successful tests.
- The test case has passed for 3 iterations with the correct argument value.

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, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_help_contains_examples` 53ms 🛡️ 3

AI ASSESSMENT

Scenario: The CLI help text should include usage examples.

Why Needed: This test prevents a bug where the help message does not contain any usage examples, potentially confusing users.

Key Assertions:

- The `--help` option is present in the help output.
- The `*Example:*--llm-report*` line matches the expected format of an example.
- The test checks that the help text includes a clear and concise description of how to use the plugin.

COVERAGE

src/pytest_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, 387-388, 391, 395-397)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_markers_registered` 49ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that LLM markers are registered and their presence is detected by pytester.

Why Needed: Prevents a potential bug where the LLM marker registration test fails without detecting the markers.

Key Assertions:

- The `pytester` instance runs the `--markers` flag with the expected output.
- The `stdout.fnmatch_lines` method is called with the expected list of strings.
- Each string in the list matches one of the expected marker names.

COVERAGE

src/pytest_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, 387-388, 391, 395-397)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_plugin_registered` 55ms 🛡️ 3

AI ASSESSMENT

Scenario: The plugin registered with pytest11 should be visible in the help output.

Why Needed: This test prevents a potential issue where the plugin is not listed in the help message.

Key Assertions:

- The 'llm-report' option should be present in the help output.
- The 'llm-report' option should be followed by any other options.
- The 'llm-report' option should be preceded by a '--help' command.
- The plugin name should be visible after listing all plugins.
- The plugin name should not be hidden behind long list of options.
- The help output should contain the correct information about the plugin.
- The help output should display 'llm-report' option 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`	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, 387-388, 391, 395-397)

PASSED `tests/test_smoke_pytester.py::TestSpecialCharacters::test_special_chars_in_nodeid` 96ms 🛡️ 7

AI ASSESSMENT

Scenario: Test that special characters in nodeid are handled correctly.

Why Needed: This test prevents a potential bug where special characters in the nodeid field could cause issues with the PyTorch Lightning model.

Key Assertions:

- The `s` parameter passed to the `test_special_chars_in_nodeid` function should not be empty.
- The HTML report generated by PyTorch Lightning should contain the `` tag.
- The nodeid field in the report file should not cause any errors or crashes.
- The contents of the report file should be valid and consistent with the expected format.
- The `report_path.exists()` assertion should pass, indicating that the test did not crash.
- The `content` variable read from the report path should contain the `` tag as expected.

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, 331, 337-338, 365-

	375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
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 🛡️ 3

AI ASSESSMENT

Scenario: Tests the function `format_duration` with a boundary of exactly one minute.

Why Needed: This test prevents regression in the `format_duration` function when input is exactly one minute.

Key Assertions:

- The result should be '1m 0.0s' (one minute and zero seconds).
- The time unit should be 'm' (minutes) instead of 's' (seconds).
- The seconds part should be zero.
- The function should handle input exactly one minute without any errors.
- The function should return the correct format string for one minute.
- The function should not raise an exception when given a negative time 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: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

PASSED `tests/test_time.py::TestFormatDuration::test_microseconds_format` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the test_time module's format_duration function with a sub-millisecond duration.

Why Needed: Prevents regression or bug that may occur when using durations greater than one millisecond.

Key Assertions:

- The result of `format_duration(0.0005)` should contain 'µs' in its string representation.
- The result of `format_duration(0.0005)` should be equal to '500µs'.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	2 lines (ranges: 39-40)

PASSED `tests/test_time.py::TestFormatDuration::test_milliseconds_format` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test formats sub-second durations as milliseconds.

Why Needed: Prevents regression where the test fails due to incorrect formatting of millisecond durations.

Key Assertions:

- The function `format_duration(0.5)` returns a string in the format 'X.XXms' where X is the number of milliseconds.
- The assertion `assert result == '500.0ms'` checks if the formatted string matches the expected output.
- The assertion `assert 'ms' in result` verifies that the string contains the substring 'ms'.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/time.py`	3 lines (ranges: 39, 41-42)

PASSED `tests/test_time.py::TestFormatDuration::test_minutes_format` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `format_duration` function correctly formats durations over a minute.

Why Needed: This test prevents regression when the duration is greater than one minute, as it should be displayed in minutes and seconds format.

Key Assertions:

- The result contains 'm' (minutes) and 's' (seconds)
- The result equals '1m 30.5s'
- The function correctly handles durations over a minute
- The function displays the duration in minutes and seconds format

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/time.py`	6 lines (ranges: 39, 41, 43, 46-48)

PASSED `tests/test_time.py::TestFormatDuration::test_multiple_minutes` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 3 minutes and 5 seconds.

Why Needed: This test prevents regression in handling durations that include fractional parts (e.g., 2 hours 30 minutes).

Key Assertions:

- The result is '3m 5.0s' as expected.
- The duration value is correctly calculated to be 185 seconds.
- The function handles values with fractional parts without truncation or rounding 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/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/time.py`	`6 lines (ranges: 39, 41, 43, 46-48)`

PASSED `tests/test_time.py::TestFormatDuration::test_one_second` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a single-second input.

Why Needed: Prevents regression in formatting durations to seconds.

Key Assertions:

- The output of `format_duration(1.0)` should be '1.00s'.
- The duration is formatted as '1.00s' instead of just '1s'.
- The function correctly handles inputs greater than zero.
- The function does not raise an error for invalid input (e.g., negative numbers).
- The formatting is consistent across different Python versions and platforms.
- The test covers a specific edge case (one second) to ensure robustness.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED `tests/test_time.py::TestFormatDuration::test_seconds_format` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the `format_duration` function to ensure it correctly formats seconds under a minute.

Why Needed: This test prevents regression when the duration is less than one minute, as the current implementation does not handle this case.

Key Assertions:

- The result contains the string 's' (for seconds) in its format.
- The result equals the expected string '5.50s'.
- The function correctly handles durations under a minute without any additional formatting.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/time.py`	4 lines (ranges: 39, 41, 43-44)

PASSED tests/test_time.py::TestFormatDuration::test_small_milliseconds 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 1 millisecond.

Why Needed: This test prevents a potential issue where the function incorrectly formats durations in milliseconds.

Key Assertions:

- The output should be '1.0ms' for a duration of 1 millisecond.
- The function should handle cases where the input is exactly 1 millisecond without any rounding errors.
- The function should not round up to a larger value than the input when it's already an integer (e.g., 0.999... becomes 1.0ms).
- The function should correctly format durations in milliseconds, including decimal points and commas.
- The function should handle negative durations correctly.
- The function should not silently truncate or round up to a larger value when the input is already an integer (e.g., -1.999... becomes -2.0ms).
- The function should raise an error if the input duration is not a non-negative number.

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	3 lines (ranges: 39, 41-42)

PASSED `tests/test_time.py::TestFormatDuration::test_very_small_microseconds` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `format_duration` function correctly formats very small durations as microseconds.

Why Needed: This test prevents a potential bug where the function does not handle very small durations correctly, potentially leading to incorrect formatting.

Key Assertions:

- The result of calling `format_duration(0.000001)` should be '1µs'.
- The string representation of the result should match '1µs'.
- The unit suffix ('µ') is present in the result.
- The function correctly handles very small durations (less than 10^-6 seconds).
- The function does not raise an exception when given a duration less than 10^-9 seconds.
- The function returns '1' as expected for a duration of exactly 0.000001 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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/time.py`	`2 lines (ranges: 39-40)`

PASSED `tests/test_time.py::TestIsoFormat::test_formats_datetime_with_utc` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the functionality of formatting datetime objects with UTC timezone.

Why Needed: This test prevents regressions where datetime objects with UTC timezone are not correctly formatted as ISO format.

Key Assertions:

- The function `iso_format(dt)` returns a string in the correct ISO format for datetime objects with UTC timezone.
- The function `iso_format(dt)` handles cases where the input datetime object has an invalid timezone.
- The function `iso_format(dt)` correctly formats datetime objects with UTC timezone into the expected ISO format.
- The function `iso_format(dt)` raises an error when given a datetime object without a valid timezone.
- The function `iso_format(dt)` handles cases where the input datetime object is in a timezone other than UTC.
- The function `iso_format(dt)` correctly formats datetime objects with UTC timezone into the correct ISO format.
- The function `iso_format(dt)` returns an error message when given a datetime object without a valid timezone.

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/time.py`	1 lines (ranges: 27)

PASSED `tests/test_time.py::TestIsoFormat::test_formats_naive_datetime` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test formats naive datetime with no timezone.

Why Needed: Prevents regression in naive datetime format handling without timezone.

Key Assertions:

- The function `iso_format(dt)` should return the correct ISO formatted string for a naive datetime (no timezone).
- The function `iso_format(dt)` should handle the date part correctly (2024-06-20).
- The function `iso_format(dt)` should handle the time part correctly (14:00:00).

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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/time.py`	1 lines (ranges: 27)

PASSED `tests/test_time.py::TestIsoFormat::test_formats_with_microseconds` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `iso_format` function correctly formats a datetime object with microseconds.

Why Needed: This test prevents potential issues where microseconds are not properly formatted in ISO format.

Key Assertions:

- The output of `iso_format(dt)` should contain the string '123456' which represents the microseconds part of the datetime.
- The microseconds value should be a four-digit number (e.g., '0000', '1000', etc.).
- Any non-numeric characters in the microseconds string should be ignored or removed.
- The microseconds value should not exceed 15 digits (i.e., up to 9999).
- The microseconds value should start with a leading zero if it is less than 10.
- Non-zero microseconds values should only contain numeric characters and/or underscores.
- Any non-numeric characters in the microseconds string except for underscores should be ignored or removed.
- The microseconds value should not include any decimal points (e.g., '123.456', etc.).
- The microseconds value should not exceed 15 digits (i.e., up to 9999).

COVERAGE

`src/pytest_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: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/time.py`	`1 lines (ranges: 27)`

PASSED `tests/test_time.py::TestUtcNow::test_has_utc_timezone` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `utc_now()` function returns a datetime object with an associated UTC timezone.

Why Needed: Prevents regression in tests that rely on the current system's timezone being set to UTC.

Key Assertions:

- The returned datetime object has a valid timezone information (tzinfo attribute is not None and equals `UTC`).
- The returned datetime object's timezone is indeed 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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/time.py`	1 lines (ranges: 15)

PASSED `tests/test_time.py::TestUtcNow::test_is_current_time` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies the correctness of `utc_now()` function by comparing its return value with the current time.

Why Needed: This test prevents a potential issue where the `utc_now()` function returns an incorrect or outdated time due to a timing-related bug.

Key Assertions:

- The returned time should be within a certain tolerance (e.g., 1 second) of the actual current time.
- The returned time should not exceed the actual current time by more than the specified tolerance.
- The returned time should not be less than the actual current time by more than the specified tolerance.
- If the system is running slowly, the `utc_now()` function may return an outdated time.
- If the system is experiencing a timing-related issue (e.g., network lag), the `utc_now()` function may return an incorrect or outdated 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: 387-388, 391, 395-397)
`src/pytest_llm_report/util/time.py`	1 lines (ranges: 15)

PASSED `tests/test_time.py::TestUtcNow::test_returns_datetime` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The function `utc_now()` returns a `datetime` object.

Why Needed: This test prevents the regression of returning an incorrect or invalid datetime when calling `utc_now()`.

Key Assertions:

- result is an instance of `datetime`
- result has a valid timezone.
- result does not raise any exceptions
- result is not `None`
- result is not a string
- result is not a timedelta

COVERAGE

src/pytest_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: 387-388, 391, 395-397)
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/aggregation.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, 196, 205, 217, 219-233, 235,	66, 90-91, 192, 203

					237, 245-246, 248-249, 251, 253-255, 259, 262-263, 265-266, 269-271, 273, 275-276, 280	
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	74
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

					104, 114, 116-117, 127, 132, 134-135	
src/pytest_llm_report/llm/schemas.py	36	1	35	97.22%	8, 10-12, 16, 22, 38, 42-44, 46-47, 50-53, 55, 58-59, 62-65, 67-68, 77, 84, 90, 94-98, 102, 130	39
src/pytest_llm_report/models.py	240	10	230	95.83%	17-18, 21, 24-25, 34-36, 38, 40, 47-48, 61-67, 69, 71, 82-83, 95-100, 102, 104, 109-115, 118-119, 141-157, 159, 161, 167-171, 173-182, 184, 186, 188-190, 193-194, 202-203, 205, 207, 213-214, 223-225, 227, 229, 233-235, 238-239, 248-250, 252, 254, 261-262, 271-273, 275, 277, 281-283, 286-287, 324-353, 355-360, 362, 364, 382-405, 407-419, 422-423, 437-445, 447, 449, 459, 461, 464-465, 482-492, 494, 500, 502, 508-512, 514, 516, 518, 520, 522	172, 183, 185, 187, 460, 513, 515, 517, 519, 521
src/pytest_llm_report/options.py	117	45	72	61.54%	106, 146, 175, 178-180, 185-187, 193-195, 201-203, 209-218, 220, 224, 233, 248, 251-267, 270-283, 286-295, 298, 300	13-15, 21-22, 90-94, 97-99, 102-105, 122-123, 126-132, 135-137, 140-142, 145, 156-160, 163-164, 167, 169, 222, 227, 236

src/pytest_llm_report/plugin.py	156	25	131	83.97%	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, 309-314, 317-318, 322-323, 331-332, 337-340, 343, 345, 348-353, 355, 357, 365-366, 387-388, 391-392, 395-397, 408-409, 412, 415-416, 419-421, 431-432, 435-437, 448-449, 452, 455, 457-458	13, 15-17, 19-20, 22, 28-31, 34, 160, 216, 319, 327-328, 333-334, 379-380, 400, 424, 440-441
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	80, 114, 142, 146, 149
src/pytest_llm_report/render.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	
src/pytest_llm_report/report_writer.py	167	10	157	94.01%	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	113, 135-137, 424-425, 432, 449-451
src/pytest_llm_report/util/fs.py	34	3	31	91.18%	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	40, 65, 67
src/pytest_llm_report/util/hashing.py	36	0	36	100.0%	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/util/ranges.py	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	-
src/pytest_llm_report/util/time.py	16	0	16	100.0%	4, 6, 9, 15, 18, 27, 30, 39-44, 46-48	-