Test Report

Run ID: 20977294143-py3.12 • Generated: 2026-01-14 00:17:44 • Duration: 34.56s
Plugin: v0.1.0 (2f498263985a34902252c53c11fb820445bd8f21) [dirty]
Repo: v0.1.0 (2c0260c5dbd1804612183ca8a93f7af99b92662b)
LLM: ollama / llama3.2:1b (minimal context, 380 annotated, 6 errors)

92.91%
Total Coverage

387	387	0	0
TOTAL TESTS	PASSED	FAILED	SKIPPED

0	0	0
XFAILED	XPASSED	ERRORS

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

AI ASSESSMENT

Scenario: Test that aggregating all policy results in two tests being returned.

Why Needed: Prevents regression where only one test is returned when aggregating all policy results.

Key Assertions:

- The aggregated report should contain both 'test_bar' and 'test_XXX' (where XXX is the nodeid) as expected.
- Both tests in the aggregated report should have a duration of 0.1 seconds.
- The phase of each test should be 'call'.
- The outcome of each test should be 'passed'.
- The nodeid should match the one used to create the dummy reports.
- The retention policy should retain both tests in the aggregated report.
- The number of tests in the aggregated report should be 2.

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: Tests the `aggregate` method of Aggregator when a directory does not exist.

Why Needed: Prevents a potential bug where an empty aggregation result is returned due to a missing or non-existent aggregate directory.

Key Assertions:

- The `aggregate` method should raise an exception or return an error message indicating that the aggregate directory does not exist.
- The `aggregate` method should return None instead of an empty list.
- A custom error message should be provided to indicate that the aggregate directory does not exist.
- The test should verify that the `aggregate` method raises an exception when called with a non-existent directory.
- The test should verify that the `aggregate` method returns an empty list when called with a non-existent directory.
- A specific error message should be provided to indicate that the aggregate directory does not exist.
- The test should check for any custom error messages provided by the `pathlib.Path.exists` mock.

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 latest policy is selected when aggregating reports with different times but same test outcome.

Why Needed: This test prevents a regression where the latest policy might not be chosen due to conflicting report outcomes or times.

Key Assertions:

- The result of `aggregator.aggregate()` should contain only one test with an outcome of 'passed'.
- The number of tests in the result should be 1.
- The outcome of the first test in the result should match the latest policy outcome (from report2).
- The run meta should indicate that the aggregation was performed on a single run.
- The run meta should contain two runs (one for each report).
- The summary should have one passed and zero failed tests.
- The passed count should be 1, matching the latest policy outcome.

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 regression in case the test relies on a specific directory for aggregation.

Key Assertions:

- agg.aggregate() should return None if mock_config.aggregate_dir is None.
- agg.aggregate() should not raise an error or exception if mock_config.aggregate_dir is None.

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 should not return any results when there are no reports.

Why Needed: This test prevents a potential bug where the `aggregate` method returns `None` instead of raising an error or returning an empty list when there are no reports.

Key Assertions:

- The `aggregate` method is called without any arguments.
- The `aggregate` method should raise an exception or return an empty list if there are no reports.
- The `aggregate` method does not throw a `ValueError` or `NoneType` exception when there are no reports.
- The `aggregate` method returns an empty list instead of raising an error or returning an empty list.
- The `aggregate` method should raise an exception with a meaningful message if there are no reports.
- The `aggregate` method does not log any errors or warnings when there are no reports.
- The `aggregate` method does not throw any exceptions when there are multiple reports.
- The `aggregate` method returns the correct number of aggregates based on the input data.

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.

Why Needed: Prevents regression in core functionality by ensuring accurate coverage and LLM annotation data is preserved during serialization.

Key Assertions:

- Coverage was properly deserialized with correct file paths and line ranges.
- LLM annotation was properly deserialized with correct scenario, why needed, and key assertions.
- Serialized report contains correct coverage and LLM annotations.

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 source coverage summary aggregation with a single report.

Why Needed: Prevents regression where multiple reports are aggregated and the source coverage is not correctly calculated for each report.

Key Assertions:

- The `source_coverage` list contains exactly one `SourceCoverageEntry` object.
- Each `SourceCoverageEntry` has the correct `file_path`, `statements`, `missed`, `covered`, `coverage_percent`, and `covered_ranges` values.
- The `source_coverage` list is not empty.
- All `SourceCoverageEntry` objects in the list are instances of `SourceCoverageEntry`.
- Each `SourceCoverageEntry` object has a `file_path` attribute that matches the expected value.
- For each report, the source coverage summary correctly calculates the covered and missed statements.
- The source coverage percentage is calculated correctly for each report.

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 bug where the test fails due to missing or incorrect configuration of llm_coverage_source.

Key Assertions:

- Verify that aggregator._load_coverage_from_source() returns None when llm_coverage_source is not set.
- Verify that aggregator._load_coverage_from_source() raises a UserWarning when llm_coverage_source does not exist.
- Verify that aggregator._load_coverage_from_source() correctly loads coverage from the configured source file (mocking coverage.py).
- Verify that aggregator._load_coverage_from_source() returns None when the configured source file does not exist.
- Verify that aggregator._load_coverage_from_source() correctly handles mock coverage and mapper classes.
- Verify that aggregator._load_coverage_from_source() correctly calls the load method on the mock coverage class.

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 the `_recalculate_summary` method of `Aggregator` class to ensure it recalculates the latest summary correctly when new test results are added.

Why Needed: The current implementation does not handle new test results properly, which can cause incorrect calculations of the latest summary.

Key Assertions:

- Verify that the total count is updated correctly with the new test results.
- Check if the passed count remains unchanged despite new test failures.
- Ensure the skipped count is preserved and doesn't change unexpectedly.
- Verify the xfailed count is updated accordingly.
- Confirm the xpassed count does not get affected by new test failures.
- Test that the error count remains consistent with the latest summary.
- Verify the coverage percentage is correctly calculated and preserved.

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 Aggregator should skip reports containing invalid JSON files.

Why Needed: Prevents a potential bug where the test incorrectly counts all reports, including those with invalid JSON.

Key Assertions:

- The `aggregate` function correctly identifies and skips reports with invalid JSON files.
- Only valid report files are counted in the aggregate result.
- Invalid JSON file is marked as a warning instead of skipping it.

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 correctly recalculates the summary coverage when new tests are added and the latest summary is updated.

Why Needed: This test prevents a regression where the aggregator's recalculate_summary method fails to update the summary coverage when new tests are added, potentially leading to inaccurate reporting of coverage.

Key Assertions:

- The total number of nodes passed in the latest summary should be equal to the number of tests that failed.
- The total duration of all nodes passed in the latest summary should be greater than or equal to the sum of durations of all nodes failed in the latest summary.
- The coverage total percent of all nodes passed in the latest summary should be greater than or equal to 88.5%
- All nodes passed in the latest summary should have a duration greater than zero.
- All nodes failed in the latest summary should have a duration less than or equal to the sum of durations of all nodes passed in the latest summary.

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: The `test_cached_tests_are_skipped` function verifies that cached tests are skipped by the annotator.

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

Key Assertions:

- The `mock_provider`, `mock_cache`, and `mock_assembler` mocks should be created with `None` as their arguments.
- The `test_cached_tests_are_skipped` function should not call any methods on these mocks.
- Any method calls on the mocks should result in a `AttributeError` or a similar exception.
- The `test_cached_tests_are_skipped` function should not modify the test results.
- The `test_cached_tests_are_skipped` function should only return `None` as its result.
- The `mock_provider`, `mock_cache`, and `mock_assembler` mocks should be created with a valid cache directory.
- Any exceptions raised by the `mock_provider`, `mock_cache`, or `mock_assembler` methods should be caught and logged correctly.

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: Verify concurrent annotation functionality in TestAnnotateTests

Why Needed: Prevents potential performance degradation due to concurrent annotation requests.

Key Assertions:

- Ensures that the annotator does not exceed the maximum allowed concurrency limit when multiple annotations are performed simultaneously.
- Verifies that the annotator properly handles concurrent requests without causing a deadlock or other unexpected behavior.
- Checks if the annotator correctly updates its internal state to reflect concurrent annotation requests.
- Ensures that the annotator does not introduce any significant performance degradation due to excessive concurrency.
- Verify that the annotator properly handles cases where multiple annotations are performed concurrently with no progress made.
- Checks for any potential issues related to thread safety or synchronization when performing concurrent annotations.
- Verifies that the annotator correctly handles edge cases such as concurrent annotation requests with no available resources.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	64 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104-112, 129-135, 137, 139, 229-232, 234, 236-237, 239, 245-246, 248-253, 255, 261, 266)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The annotator handles failures when multiple annotations are performed concurrently.

Why Needed: This test prevents the annotator from crashing due to concurrent annotation failures, ensuring it remains stable and functional.

Key Assertions:

- mock_provider is not called with a failure message.
- mock_cache is not modified by the annotation process.
- mock_assembler does not raise an exception when handling failures.

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 verifies the progress reporting functionality of the annotator.

Why Needed: This test prevents regressions that may occur when the progress reporting feature is not implemented or is disabled.

Key Assertions:

- Verify that the progress bar updates correctly as annotations are added.
- Check if the progress bar displays a meaningful percentage value.
- Ensure that the progress bar resets to zero after all annotations have been processed.
- Verify that the progress bar increments correctly when new annotations are added.
- Test if the progress bar reflects the actual time elapsed since the last update.
- Confirm that the progress bar does not freeze or become unresponsive during processing.

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 sequential annotation functionality in the annotate function

Why Needed: Prevents regression due to incorrect sequential annotation behavior when multiple annotators are used.

Key Assertions:

- The `annotate` function should correctly apply annotations sequentially to a document.
- Multiple annotators should be able to work together without causing any errors or inconsistencies.
- The `annotate` function should handle overlapping annotations correctly and produce the expected output.
- The `annotate` function should not introduce any new bugs or regressions when using multiple annotators.
- The `annotate` function should maintain its performance and scalability even with a large number of annotators.
- The `annotate` function should be able to handle complex annotation tasks without causing any errors or inconsistencies.

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 verifies that the annotator does not perform any actions when the LLM (Large Language Model) is disabled.

Why Needed: This test prevents a regression where the annotator would incorrectly skip tests or annotations in such a scenario.

Key Assertions:

- A configuration object with 'provider' set to 'none' is created.
- The annotate_tests function is called with an empty list of tests and this configuration.
- No annotation results are produced for the annotated tests.
- The LLM is disabled before any annotation process can occur.
- The annotator does not perform any actions when the LLM is disabled.
- No errors or exceptions are raised during the execution of the annotate_tests function.
- The test passes without any failures or warnings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 should skip the annotation process when a provider is unavailable.

Why Needed: This test prevents regression by ensuring that annotators do not attempt to annotate data from an unavailable provider.

Key Assertions:

- mock_provider.unavailable.assert_called_once_with('provider_name')
- self.assertEqual(mock_provider, mock_provider), 'MockProvider instance is not being created 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/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_annotate_concurrent_with_progress_and_errors verifies that the annotator reports progress and first error when annotated concurrently with errors.

Why Needed: This test prevents regression by ensuring that the annotator correctly handles concurrent annotations with errors.

Key Assertions:

- Verify that the annotator reports the correct number of tasks (2) and failures (1)
- Verify that the first error message contains 'first error'
- Verify that there are messages indicating progress ('Processing X test(s)')
- Verify that the LLM annotation is included in the failure message
- Verify that the annotator correctly handles concurrent annotations with errors

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 where the annotator does not wait for the rate limit interval to pass before proceeding with annotation tasks.

Key Assertions:

- assert mock_sleep.called
- assert mock_time.call_count == 5
- assert mock_sleep.args[0] == 1.0
- assert mock_sleep.args[1] == 100.0
- assert mock_sleep.args[2] == 100.1
- assert mock_sleep.args[3] == 100.2
- assert mock_sleep.args[4] == 100.3

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/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: Should report progress for cached tests when annotating with maximal configuration.

Why Needed: Prevents regression and ensures that cached tests are properly reported as completed.

Key Assertions:

- The `progress_msgs` list should contain any messages indicating that a test was annotated in the cache.
- Any message containing '(cache): test_cached' should be present in the `progress_msgs` list.
- If no progress messages are found, it indicates that all tests were not cached or annotated successfully.

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: Verify that the annotator throws an error when the test provider is not available.

Why Needed: This test prevents a potential regression where the annotator fails to report tests due to a missing or unavailable test provider.

Key Assertions:

- mocks.is_available was called with False, but it should have been called with True (expected)
- mock_provider.get_provider was called, and it returned mock_provider (actual)
- annotate_tests was called with [tests], and it did not return any results (expected)
- captured.out contained the string 'not available. Skipping annotations' (expected)
- assert captured.out.contains('not available. Skipping annotations') was called with True (expected)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	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: Test that extracting malformed JSON from a response prevents the 'Failed to parse LLM response as JSON' error.

Why Needed: This test ensures that the extract_json_from_response function does not throw an exception when encountering invalid JSON in the response.

Key Assertions:

- The function `extract_json_from_response` will raise a `JSONDecodeError` with the message 'Failed to parse LLM response as JSON' when given an invalid JSON string.
- The function will not attempt to extract JSON from a malformed input.
- The error message provided by the function will accurately reflect the problem encountered during parsing.
- The function's behavior will be consistent across different test runs with varying input data.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: Tests for non-string fields in response data.

Why Needed: Prevents regression when dealing with non-string fields in the response data.

Key Assertions:

- The scenario number should be an integer.
- The why_needed list should include '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 valid instance of `GeminiProvider`.

Why Needed: This test prevents regression in case the `Config` class is not properly initialized or if the provider type is changed to 'gemini'.

Key Assertions:

- The returned value is an instance of `GeminiProvider`.
- The return value has a valid type.
- The return value does not throw any exceptions.
- The return value is not None.
- The return value can be converted to a string using the `__str__()` method.
- The return value can be compared to another instance of `GeminiProvider` using the `==` operator.
- The return value has the correct attributes (e.g., `name`, `description`, etc.)
- The return value is not an instance of a custom class or type

COVERAGE

src/pytest_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_provider` function with an invalid provider.

Why Needed: To prevent a ValueError exception from being raised when using an unknown LLM provider.

Key Assertions:

- The `get_provider` function should raise a ValueError with the message 'Unknown LLM provider: invalid'.
- The error message should contain the string 'invalid'.
- The error message should not be empty.
- The error message should not be a single line of text.
- The error message should be in English.
- The error message should not be a generic error message like '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 an instance of `LiteLLMProvider` when a valid provider is provided.

Why Needed: This test prevents a potential bug where the `get_litellm_provider` function might return an incorrect or invalid object if no valid provider is provided.

Key Assertions:

- The returned value should be an instance of `LiteLLMProvider`.
- The returned value should have the correct attributes (e.g. `provider`, `name`, etc.).
- No exceptions should be raised when a valid provider is provided.
- An exception should be raised if no valid provider is provided.
- The function name should match the expected return type (`get_litellm_provider`)
- The function signature should match the expected input parameters (e.g. `config`)
- The function's docstring should contain information about its purpose and 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/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: Verifies that `get_provider` returns a `NoopProvider` instance when the `provider` is set to 'none'.

Why Needed: Prevents a potential bug where `get_provider` throws an exception or raises an error when the provider is not specified.

Key Assertions:

- The function `get_provider(config)` returns an instance of `NoopProvider`.
- The `provider` parameter passed to `get_provider` has no effect on the return value.
- A `NoopProvider` instance is created successfully without any errors or exceptions.
- The `Config` object passed to `get_provider` does not affect the behavior of the function.
- The `None` provider is recognized and used correctly by the function.
- The function handles cases where no provider is specified correctly.
- The return value of `get_provider` matches the expected type `NoopProvider`.
- A `NoopProvider` instance is created without any additional arguments or 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/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: Verify that the `get_ollama_provider` function returns an instance of `OllamaProvider` when a valid configuration is provided.

Why Needed: This test prevents regression in case the `Config` class or `get_provider` function changes, potentially introducing issues with provider selection.

Key Assertions:

- The returned value should be an instance of `OllamaProvider`.
- The `provider` attribute of the returned value should have a value that matches the expected `ollama` configuration.
- The `get_provider` function should be able to successfully retrieve an Ollama provider with the provided configuration.
- The `Config` class should be able to create a valid configuration object for the `ollama` provider.
- The `get_provider` function should be able to return an instance of `OllamaProvider` when given a valid configuration.
- The `provider` attribute of the returned instance should have the correct value.
- The `provider` attribute of the returned instance should not raise any exceptions or 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	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: Test that a provider with an available cache is correctly identified as such.

Why Needed: This test prevents regression in cases where the LLM provider is not properly configured to use an available cache.

Key Assertions:

- The `is_available()` method returns True for the provided provider.
- The `is_available()` method returns True for another instance of the same provider.
- The `checks` attribute of the provider is set to 1 after calling `_check_availability()`.
- _check_availability() should increment the `checks` counter and return True.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: When the `get_model_name` method of a `ConcreteProvider` instance is called with a default configuration, then the model name returned should be 'test-model'.

Why Needed: This test prevents a potential regression where the model name defaults to the configuration without any explicit override.

Key Assertions:

- The `model` attribute of the `Config` object passed to `get_model_name()` is set to 'test-model'.
- The `provider` instance has a `get_model_name()` method that returns 'test-model'.
- The `provider.get_model_name()` call does not raise an exception or return any error.
- The `model_name` attribute of the returned model object is equal to 'test-model'.
- The `model_name` attribute of the returned model object is set to a value other than 'test-model'.
- The `provider.get_model_name()` method does not raise an exception when called with an invalid configuration.
- The `provider.get_model_name()` method returns an incorrect model name for some configurations.
- The `model_name` attribute of the returned model object is set to a value that is different from 'test-model' for all 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/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: The test verifies that the `get_rate_limits` method of a `ConcreteProvider` instance returns `None` when no default rate limits are specified.

Why Needed: This test prevents a potential regression where the default rate limits are not properly set for a provider without any explicit configuration.

Key Assertions:

- The `rate_limits` attribute is `None`.
- The `get_rate_limits()` method returns `None`.
- A provider with no specified default rate limits raises an exception when trying to get the rate limits.
- The `rate_limits` value is not explicitly set in the configuration.
- The `ConcreteProvider` class does not have a `get_rate_limits()` method by default.
- The `Config` class has a `DEFAULT_RATE_LIMITS` attribute that can be used to specify default rate limits.
- Setting `DEFAULT_RATE_LIMITS` to an empty list or dictionary allows the provider to use the specified defaults.
- Using `None` as the value for `DEFAULT_RATE_LIMITS` in the configuration will prevent the test from failing.

COVERAGE

src/pytest_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 `is_local()` returns False when the default is set to false.

Why Needed: Prevents regression in case the default is changed to false.

Key Assertions:

- The `provider.is_local()` method should return `False` when the default value is `False`.
- If the default value is `True`, the method should return `True`.
- If the default value is an empty string, the method should raise a `ValueError`.
- If the default value is not a boolean, the method should raise a `TypeError`.
- The method should be able to handle different types of default values (e.g. integers, strings).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 consistent.

Why Needed: This test prevents a bug where different sources of the same function could produce different hashes, potentially leading to unexpected behavior or errors.

Key Assertions:

- The hash of `source` should be equal to itself.
- The hash of `source` should not change if the source code remains the same.
- If the source code is modified and then re-run the test, the hash should remain unchanged.

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: Verifies that the `hash_source` function returns a different hash for two different source functions.

Why Needed: Prevents a potential bug where the same source code produces the same hash value, potentially leading to incorrect caching behavior.

Key Assertions:

- The output of `hash_source('def test_a(): pass')` should be different from `hash_source('def test_b(): pass')`.
- The output of `hash_source('def test_c(): pass')` should not match either of the above outputs.
- The hash value returned by `hash_source()` for a given source code should always be unique.
- If two different source functions produce the same hash, it may indicate an issue with caching or data integrity.
- Inconsistent hashes can lead to incorrect caching behavior, potentially causing unexpected results in the application.

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: Verify the length of the generated hash.

Why Needed: Prevents a potential issue where the hash may not be long enough to cover all possible inputs.

Key Assertions:

- The hash should have a length of 16 characters.
- The hash is at least 16 characters long.
- The hash does not exceed 15 characters.
- The hash is not empty.
- The hash contains only ASCII letters, digits, and underscores.
- The hash does not contain any non-ASCII characters.
- The hash has a leading zero.
- The hash does not have trailing zeros.

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: The test verifies that the `clear` method of `LlmCache` removes all cache entries.

Why Needed: This test prevents a potential bug where cache entries are not properly cleared when the cache directory is deleted or renamed.

Key Assertions:

- The cache should have exactly two entries after clearing.
- All cache entries for 'test::a' and 'test::b' should be removed from the cache.
- If an entry with a hash of 'hash1' exists in the cache, it should be removed.
- If an entry with a hash of 'hash2' exists in the cache, it should also be removed.
- The `get` method for 'test::a' and 'test::b' should return None if their respective hashes are not found in the cache.

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 are not cached.

Why Needed: To prevent caching of error-related annotations, ensuring that only valid annotations are stored in the cache.

Key Assertions:

- The annotation 'error' should be present in the cache.
- The value associated with 'test::foo' should be 'abc123'.
- The annotation is not cached for the given key.
- The value of the annotation is not 'abc123'.
- A non-error annotation ('API timeout') is stored in the cache.

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: Verify that the `get` method returns `None` for missing entries in the cache.

Why Needed: Prevents a potential bug where the test fails due to an incorrect assumption about the cache's behavior when no entry exists.

Key Assertions:

- The function `cache.get('test::foo', 'abc123')` should return `None`.
- The function `cache.get('test::bar', 'xyz789')` should also return `None`.
- The cache's internal state should not be modified unexpectedly when no entry exists.
- The test should pass even if the cache is initialized with a different configuration than expected.
- The test should handle missing keys correctly, without raising an exception or throwing an error.
- The function should return `None` for all other non-existent keys in the cache.
- The `tmp_path / 'cache'` path creation should be able to create the correct directory structure and permissions.

COVERAGE

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

Why Needed: This test prevents bypass attacks by ensuring that annotations are stored and retrieved correctly.

Key Assertions:

- Verify the correct storage of annotations in the cache.
- Check that the retrieved annotation matches the expected scenario.
- Ensure the confidence level of the retrieved annotation is as expected.

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: Test verifies that collection errors have the correct structure.

Why Needed: This test prevents a potential bug where the structure of collection errors is not checked, potentially leading to incorrect handling or reporting.

Key Assertions:

- The 'nodeid' attribute of the error object matches the expected value.
- The 'message' attribute of the error object matches the expected value.
- The 'nodeid' and 'message' attributes are present in the error object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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: Test verifies that an empty collection is returned when the `get_collection_errors` method is called initially.

Why Needed: This test prevents a potential regression where an empty collection is not immediately recognized as such, potentially causing downstream issues.

Key Assertions:

- The `collector.get_collection_errors()` method should return an empty list.
- A call to `collector.get_collection_errors()` before the first error occurs should raise an exception or indicate that no errors are present.
- The test should fail if the collection is not empty after calling `get_collection_errors`.
- The test should pass if the collection is empty after calling `get_collection_errors`.

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: Test 'test_llm_context_override_default_none' verifies that the default llm_context_override is set to None.

Why Needed: This test prevents a potential bug where the default llm_context_override is not set correctly, potentially leading to incorrect results or unexpected behavior.

Key Assertions:

- The llm_context_override attribute of TestCaseResult node 'test.py::test_foo' should be None.
- The llm_context_override value should match the expected default value for the given context.
- The llm_context_override attribute should not be set to a different value than the default.
- The test result should indicate that the default llm_context_override is correctly set to None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 value of llm_opt_out is set to False for a test case.

Why Needed: This test prevents regression where the default value of llm_opt_out might be incorrectly set to True for a test case with an optional parameter.

Key Assertions:

- The `llm_opt_out` attribute is not set to `True`.
- The `llm_opt_out` attribute is set to `False`.
- The value of `llm_opt_out` does not match the expected default value for this test case.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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 `capture` feature is not enabled by default.

Why Needed: This test prevents a bug where the output capture feature is disabled by default, potentially leading to unexpected behavior or errors.

Key Assertions:

- config.capture_failed_output should be False
- output_capture_enabled should be False
- capture_output_format should not be set to 'failed'

COVERAGE

src/pytest_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 set to 4000.

Why Needed: This test prevents a potential bug where the default max chars is not set correctly, potentially leading to unexpected behavior or errors when capturing output.

Key Assertions:

- config.capture_output_max_chars
- assert config.capture_output_max_chars == 4000

COVERAGE

src/pytest_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 recorded as xfailed in the TestCollector.

Why Needed: This test prevents regression where xfail failures are not properly recorded as xfailed, potentially leading to incorrect reporting of test results.

Key Assertions:

- The `results` dictionary should contain an entry for the failed node with key 'xfailed'.
- The `outcome` attribute of the failed node's result should be set to 'xfailed'.
- The `wasxfail` attribute of the failed node's result should be set to 'expected failure'.
- The duration of the test run should not affect the outcome of the xfail.
- The longrepr attribute of the failed report should contain an assertion error message.
- The wasxfail key in the failed report should match the expected failure message.
- The nodeid of the failed report matches the expected failure location.
- The when and passed attributes of the failed report are set to their correct values.

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

Why Needed: This test prevents regression where an expected failure is incorrectly marked as passed due to a bug in the collector.

Key Assertions:

- The `result.outcome` of the `test_unexpected_pass` node should be 'xpassed' when an xfail is passed.
- The `wasxfail` field of the `report` should contain 'expected failure'.
- The `duration` and `longrepr` fields of the `report` should not have any impact on the outcome.
- The `skipped` field of the `report` should be False when an xfail is passed.
- The `failed` field of the `report` should be False when an xfail is passed.
- The `passed` field of the `report` should be True when an xfail is passed.

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 that the `create_collector` function initializes correctly and returns an empty collection.

Why Needed: This test prevents a potential bug where the collector is not initialized properly, leading to incorrect results or errors.

Key Assertions:

- The `results` attribute of the `collector` object should be an empty dictionary.
- The `collection_errors` attribute of the `collector` object should be an empty list.
- The `collected_count` attribute of the `collector` object should be 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: Test the `get_results` method of `TestCollector` to ensure it returns sorted results by nodeid.

Why Needed: This test prevents a regression where unsorted results are returned, potentially affecting downstream analysis or reporting.

Key Assertions:

- The list of nodeids in the results should be in ascending order.
- The list of nodeids in the results should contain only the specified nodes (`'a_test.py::test_a'` and `'z_test.py::test_z'`).
- The `nodeid` attribute of each result object is correctly set to its corresponding value from the input data.
- No duplicate nodeids are present in the results.
- The order of nodeids is preserved even if there are multiple instances of the same nodeid (e.g., multiple tests with the same outcome).
- The `outcome` attribute of each result object matches its expected value based on the input data.
- No unexpected or missing attributes are present in the results.

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 potential bugs where an item is not properly tracked as collected or deselected, leading to incorrect counts in the `collected_count` and `deselected_count` attributes.

Key Assertions:

- The `collected_count` attribute should be equal to the number of items that were collected (3) after calling `handle_collection_finish`.
- The `deselected_count` attribute should be equal to the number of deselected items (1) after calling `handle_collection_finish`.
- The `collected_count` and `deselected_count` attributes should match the expected values before and after calling `handle_collection_finish` for all items.

COVERAGE

`src/pytest_llm_report/collector.py`	`16 lines (ranges: 78-79, 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

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

AI ASSESSMENT

Scenario: Verify that capture output is disabled when config is set to false for integration via handle_runtest_logreport.

Why Needed: This test prevents a regression where the collector captures output even if the config is set to disable capture.

Key Assertions:

- The `captured_stdout` attribute of the result should be `None` when `config.capture_failed_output` is `False` and `handle_runtest_logreport` is called with a report that has `output` as its value.
- The `wasxfail` attribute of the report should not be present in the collector's results.
- The `results` dictionary of the collector's results should contain an entry for node 't' with `captured_stdout` set to `None` and `wasxfail` set to `False`.
- The `when` value of the report should be `'call'` when it is used in the collector's handle_runtest_logreport method.
- The `passed` attribute of the report should be `True` when it is used in the collector's results.
- The `failed` attribute of the report should be `False` when it is used in the collector's results.
- The `skipped` attribute of the report should be `False` when it is used in the collector's results.
- The `capstdout` value of the report should be `'output'` when it is used in the collector's handle_runtest_logreport method.

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 `test_capture_output_stderr` method captures stderr correctly.

Why Needed: This test prevents a potential bug where the collector fails to capture stderr and reports an incorrect captured value.

Key Assertions:

- The `captured_stderr` attribute of the `TestCaseResult` object is set to 'Some error'.
- The `report.capstderr` method is called with 'Some error' as its argument.
- The `collector._capture_output(result, report)` call sets the `captured_stderr` attribute of the `TestCaseResult` object to 'Some error'.

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: Test that the `test_capture_output_stdout` function captures stdout correctly.

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

Key Assertions:

- The `captured_stdout` attribute of the `TestCaseResult` object should contain the expected output.
- The `report.capstdout` method should set the captured stdout to 'Some output'.
- The `collector._capture_output(result, report)` function should record the captured stdout correctly.
- The `result.captured_stdout` attribute should be updated with the captured stdout after calling `_capture_output`.
- The `report.capstderr` attribute should not interfere with capturing stdout.
- The `test_capture_output_stdout` function should work as expected even when `capture_failed_output` is set to `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 collector truncates output exceeding max chars.

Why Needed: Prevents a potential bug where captured output exceeds the maximum characters.

Key Assertions:

- The `captured_stdout` attribute of the `TestCaseResult` object should be equal to '1234567890' after calling `_capture_output`.
- The `capstdout` attribute is set to '123456789012345' before calling `_capture_output`.
- The captured stdout does not exceed 10 characters.
- The captured stderr remains empty.

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: Should extract markers from item with correct parameters.

Why Needed: This test prevents a potential bug where the collector does not correctly extract markers from an item, leading to incorrect results in reports.

Key Assertions:

- item.callspec.id should be set to 'param1'.
- result.param_id should be set to 'param1'.
- result.llm_opt_out should be set to True.
- result.llm_context_override should be set to 'complete'.
- result.requirements should contain ['REQ-1', 'REQ-2'].

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 that the `extract_error` method handles ReprFileLocation correctly when a crash report is used.

Why Needed: This test prevents a potential regression where the `extract_error` method crashes due to an incorrect usage of `str()` on `Report` objects containing ReprFileLocation instances.

Key Assertions:

- The `_extract_error` method should return 'Crash report' when given a `Report` object with a `longrepr` attribute that is set to a string.
- The `report.longrepr.__str__.return_value` property should be set to 'Crash report' in the test case.
- The `collector._extract_error(report)` call should return 'Crash report' as expected.

COVERAGE

src/pytest_llm_report/collector.py	22 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: This test prevents a potential regression where the error message is not returned correctly.

Key Assertions:

- The `report.longrepr` attribute should be set to the actual error message.
- The `collector._extract_error(report)` call should return the correct string value.
- The error message should match the one stored in `report.longrepr`.
- No exception should be raised when an error occurs during collection.
- The error message should not be a default value (e.g., 'Unknown error').

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 issue where the method does not handle cases where no longrepr is available, potentially leading to incorrect or missing skip reasons in reports.

Key Assertions:

- The `_extract_skip_reason` method should return `None` for `report.longrepr = None`.
- The `_extract_skip_reason` method should not raise an exception when `report.longrepr` is `None`.
- The `_extract_skip_reason` method should preserve the original value of `report.longrepr` when no longrepr is provided.
- The `_extract_skip_reason` method should not modify the original value of `report.longrepr` when no longrepr is provided.
- The `_extract_skip_reason` method should return `None` for a report with a non-`None` longrepr.
- The `_extract_skip_reason` method should preserve the original type of `report.longrepr` when no longrepr is provided.
- The `_extract_skip_reason` method should not modify the type of `report.longrepr` when no longrepr is provided.
- The `_extract_skip_reason` method should return `None` for a report with a non-`None` longrepr and an empty string as the reason.
- The `_extract_skip_reason` method should preserve the original value of `report.longrepr` when no longrepr is provided and an empty string is used as the reason.

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 that the `_extract_skip_reason` method returns a string as expected when given a `report` object with a `longrepr` attribute.

Why Needed: Prevents regression where the test fails due to an incorrect or missing implementation of the `_extract_skip_reason` method.

Key Assertions:

- The value of `report.longrepr` is set to 'Just skipped'.
- The result of calling `collector._extract_skip_reason(report)` matches the expected 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 `collectors._extract_skip_reason` correctly extracts skip message from a tuple containing file, line and message.

Why Needed: Prevents regression where the test fails due to incorrect or missing skip reason extraction.

Key Assertions:

- The 'report.longrepr' attribute of the report object is a tuple with three elements: (file, line, message).
- The value of `str(report.longrepr)` in the assertion contains the string 'Skipped for reason'.
- The `_extract_skip_reason` method of the collector correctly extracts the skip message from the tuple containing file, line and message.
- If the report object does not have a `longrepr` attribute or its value is not a tuple with three elements, the assertion will fail.

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: Test fails to handle collection report failure correctly.

Why Needed: This test prevents a bug where the collector does not record the collection error and instead returns an empty list of errors.

Key Assertions:

- The `collection_errors` attribute is populated with exactly one item containing the nodeid 'test_broken.py' and message 'SyntaxError'.
- The `nodeid` field in the first `collection_errors` item matches the expected value 'test_broken.py'.
- The `message` field in the first `collection_errors` item matches the expected value '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 in the TestCollector's behavior when handling reruns.

Key Assertions:

- The 'rerun_count' of the report should be 1 after a successful runtest.
- The final outcome of the report should be 'failed' after a failed runtest.
- The 'rerun_count' and 'final_outcome' of the collected result should match the expected values.

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 handles setup failure correctly by recording an error message.

Why Needed: This test prevents regression where the TestCollector does not handle setup failures properly, potentially leading to incorrect or misleading report data.

Key Assertions:

- res.outcome should be 'error'
- res.phase should be 'setup'
- res.error_message should match '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: TestCollectorReportHandling::test_handle_runtest_teardown_failure verifies that a failure in teardown causes an error to be recorded.

Why Needed: This test prevents a regression where the collector fails to record errors when teardown fails after a pass.

Key Assertions:

- The `teardown` report should contain an error message indicating 'Cleanup failed'.
- The `outcome` of the runtest should be set to 'error'.
- The `phase` of the runtest should be set to 'teardown'.
- The `error_message` of the runtest should be set to 'Cleanup failed'.
- The `wasxfail` attribute of the teardown report should be False.
- The `passed` and `failed` attributes of the teardown report should both be False.
- The `duration` attribute of the teardown report should be 0.1 seconds.
- The `nodeid` attribute of the teardown report should match 't::f'.

COVERAGE

src/pytest_llm_report/collector.py	38 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-115, 124, 127-128, 130, 132-133, 135-137, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the test covers edge cases in Gemini model parsing, including an empty models list and 'all' as a valid model.

Why Needed: This test prevents regression by ensuring that the gemini_model_parsing function handles these edge cases correctly.

Key Assertions:

- The function should return an empty list when given None for the `model` parameter.
- The function should return 'all' as a valid model when given 'All' as the `model` parameter.
- The function should not return any models when given an empty string or other invalid input.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/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: Verifies that the rate limiter correctly handles edge cases where there are no tokens available.

Why Needed: This test prevents a potential bug in the rate limiter, where it incorrectly allows requests to proceed when there are no tokens available.

Key Assertions:

- The next_available_in method should return 0 for both limits (50 and 60).
- The next_available_in method should not allow any requests to pass through if there are no tokens available (i.e., it should raise an exception or return a specific value indicating that the rate limiter is at capacity).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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`, as well as `RunMeta` objects, returns expected values for certain attributes.

Why Needed: This test prevents regression in coverage calculation when using boosters with LLM annotations or run meta data.

Key Assertions:

- The `coverage_percent` attribute of the `SourceCoverageEntry` object is correctly set to 50.0 after calling `to_dict()`.
- The error message returned by `LlmAnnotation.to_dict()` matches the expected string 'timeout' when an error occurs.
- The duration attribute of the `RunMeta` object is correctly set to 1.0 after calling `to_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/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: Ensures the Mapper class initializes correctly with a given configuration.

Why Needed: Prevents a potential bug where the Mapper instance does not have access to its configured settings.

Key Assertions:

- The `config` attribute of the `CoverageMapper` instance is set to the provided `Config` object.
- The `warnings` attribute of the `CoverageMapper` instance is initialized with an empty list.
- The `config` attribute is checked for equality with the original configuration object before assignment.
- The `warnings` attribute is checked for equality with an empty list after assignment.
- A warning message is not raised when creating a new `CoverageMapper` instance without providing a 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/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: Test 'tests/test_coverage_map.py::TestCoverageMapper::test_get_warnings' verifies that the function `get_warnings` returns a list of warnings.

Why Needed: This test prevents regression in case there are no warnings to report.

Key Assertions:

- The function `get_warnings()` should return an instance of `list`.
- The function `get_warnings()` should raise an exception if no warnings are found.
- The list of warnings should not be empty.
- Each warning in the list should have a type attribute matching 'warning' or 'error'.
- Each warning in the list should have a message attribute containing a string.
- Each warning in the list should have a severity level attribute matching either 'warning' or '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/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 should return empty dict when no coverage file exists.

Why Needed: Prevents regression in case of missing coverage file, ensuring accurate coverage reporting.

Key Assertions:

- config is created with a valid path for the coverage map.
- mapper is initialized with a Config instance.
- Path.exists is mocked to return False when no coverage file exists.
- glob.glob is mocked to return an empty list when no coverage file exists.
- result is compared to an empty dictionary.
- len(mapper.warnings) is greater than or equal to 1, indicating at least one 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/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` extracts node IDs for all phases when the `include_phase` parameter is set to 'all'.

Why Needed: This test prevents a regression where the coverage map might not include all phases if the `include_phase` parameter is not set to 'all'.

Key Assertions:

- The expected node ID for each phase should match the actual extracted node ID.
- All phases (setup, run, teardown) should be included in the coverage map.
- If `include_phase` is not set to 'all', the coverage map should exclude all phases.
- The coverage map should include all nodes within each phase.
- No nodes outside of the specified phase should be included in the coverage map.
- If a node ID is missing from the expected list, it should raise an error.
- The `CoverageMapper` should correctly handle cases where multiple nodes are extracted for the same phase.
- The test should pass if the `CoverageMapper` correctly extracts node IDs for all phases when `include_phase` 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/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

Scenario: Test Extract Node ID from Empty Context

Why Needed: Prevents a potential bug where the test fails due to an empty context.

Key Assertions:

- The function _extract_nodeid() should return None for an empty string.
- The function _extract_nodeid() should return None for an empty object.
- The function _extract_nodeid() should not raise an exception when given an empty 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/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: Verify that the `nodeid` extraction function correctly filters out setup phase when `include_phase=run`.

Why Needed: This test prevents a regression where the `nodeid` extraction function fails to extract node IDs for tests in the setup phase.

Key Assertions:

- The `_extract_nodeid` method of the `CoverageMapper` instance should return `None` when the input string contains 'setup'.
- The `nodeid` variable should be set to `None` after calling `_extract_nodeid`.
- The `nodeid` extraction function should correctly exclude node IDs from tests in the setup phase.
- The test should fail with an error message indicating that the `nodeid` extraction function is not applicable for the given input string.
- The test should verify that the `nodeid` extraction function returns `None` when the input string contains 'setup' and no other phases are specified.
- The `include_phase` parameter of the `CoverageMapper` instance should be set to `'run'` to filter out setup phase node IDs.

COVERAGE

src/pytest_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: Test extracts node id from run phase context.

Why Needed: Prevents a potential bug where the node id is not extracted correctly when in run phase.

Key Assertions:

- The function _extract_nodeid() of CoverageMapper class should extract the node id from the given string.
- The function _extract_nodeid() of CoverageMapper class should return the correct node id.
- The test case should fail if the extracted node id is not 'test.py::test_foo'.
- The test case should pass if the extracted node id matches '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: Test that the test_extract_contexts_full_logic function exercises all paths in _extract_contexts.

Why Needed: This test prevents a regression where the coverage of app.py is not included in the full logic coverage.

Key Assertions:

- The function should return ['test_app.py::test_one', 'test_app.py::test_two'] as expected.
- The line count for each context in 'test_app.py::test_one' should be 2 (lines 1 and 2).
- App.py should be included in the coverage of 'test_app.py::test_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/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 should handle data with no test contexts and return an empty dictionary.

Why Needed: This test prevents a regression where the coverage map does not accurately reflect the absence of test contexts in the codebase.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	14 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 144-146)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the `CoverageMapper` correctly extracts node IDs for different phases and contexts.

Why Needed: This test prevents a bug where the mapper fails to extract node IDs for certain contexts or phases, leading to incorrect coverage reporting.

Key Assertions:

- When the `_extract_nodeid` method is called with a context that does not contain a pipe (`|`), it should return the original context.
- The `CoverageMapper` should be able to handle contexts without pipes by ignoring them and returning the original context.
- If the context contains a phase but no pipe, the mapper should ignore the phase and return the original context.
- The `CoverageMapper` should not fail to extract node IDs for contexts that contain a run phase but no pipe.
- When the `_extract_nodeid` method is called with a context that contains both phases (`setup` and `run`), it should return the correct node ID.
- If the context contains multiple phases, the mapper should ignore all phases except the one specified in the include_phase parameter.
- The `CoverageMapper` should be able to handle contexts without pipes or phases by ignoring them and returning the original context.
- When the `_extract_nodeid` method is called with a context that does not contain a pipe (`|`) and no phases, it should return the original 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/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 the case where no coverage files exist.

Why Needed: Prevent a potential warning when no coverage data is available.

Key Assertions:

- The function _load_coverage_data() should return None if no .coverage files are found.
- The number of warnings should be 1.
- The first warning code should be 'W001'.

COVERAGE

src/pytest_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 loads coverage data from a corrupt .coverage file, verifying that it raises an error and provides warnings.

Why Needed: This test prevents the regression of coverage data loading errors in case of corrupted files.

Key Assertions:

- The function _load_coverage_data() returns None when the coverage file is corrupt.
- Any warning messages contain 'Failed to read coverage data' or similar error messages.
- The warnings are not empty, indicating that an error occurred during loading.

COVERAGE

src/pytest_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, which can lead to incorrect coverage data if not properly updated by the CoverageMapper.

Key Assertions:

- The `update` method of `CoverageData` should be called at least twice for each mock instance.
- The `update` method of `CoverageData` should be called only once after mocking different instances of the `CoverageData` class.
- The `update` method of `CoverageData` should not be called before mocking any instances.
- The `update` method of `CoverageData` should not be called multiple times for the same mock instance.
- The `update` method of `CoverageData` should call the internal methods correctly (e.g., `__setstate__`, `__getstate__`) after being called by the `side_effect` 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/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: The test verifies that the `map_coverage` method returns an empty dictionary when `_load_coverage_data` returns `None`.

Why Needed: This test prevents a potential bug where the coverage map is not generated correctly if no data is loaded.

Key Assertions:

- The function `_load_coverage_data` should return None for invalid input.
- The `map_coverage` method should return an empty dictionary when no data is available.
- The test should fail when `_load_coverage_data` returns `None` and the `map_coverage` method tries to access its attributes.
- The expected result of `map_coverage` should be an empty dictionary.
- The function `map_coverage` should not raise any exceptions when called with a None value for `_load_coverage_data`.
- The test should pass only if no data is loaded into the coverage map.
- The test should fail only when no data is loaded into the coverage map.

COVERAGE

src/pytest_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 that the CoverageMapperMaximal class handles errors during analysis correctly.

Why Needed: This test prevents a regression where the CoverageMapperMaximal class fails to handle errors during coverage analysis.

Key Assertions:

- The mock CovarianceMapper instance should be able to raise an exception when called with an invalid configuration.
- The mock CovarianceMapper instance should return an empty list of entries for files that cause an error during analysis.
- The mock CovarianceMapper instance should not attempt to analyze the data if it is already analyzed (i.e., no errors occurred).
- The mock CovarianceMapper instance should raise an exception when called with a Config object that has a missing required parameter.
- The mock CovarianceMapper instance should return an empty list of entries for files that are not measured by the mocked CovarianceMapper instance.
- The mock CovarianceMapper instance should not attempt to analyze data from a mocked CovarianceMapper instance if it is already analyzed (i.e., no errors occurred).
- The mock CovarianceMapper instance should raise an exception when called with a Config object that has an invalid configuration parameter.

COVERAGE

src/pytest_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 covers all paths in map_source_coverage with comprehensive coverage.

Why Needed: This test prevents regression by ensuring that all possible source code paths are covered under the given configuration.

Key Assertions:

- The mock_cov.get_data() call should return a MagicMock object with 'app.py' as the returned value.
- The mock_cov.analysis2() call should return a tuple containing 'app.py', [1, 2, 3], [], [2], [].
- All statements in the entries list should be equal to 3.
- All covered files should have a coverage percentage of 66.67%.
- The length of the entries list 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/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 with a valid warning code and message.

Why Needed: Prevents a potential bug where an unknown or invalid warning code is passed to the `make_warning` function, causing it to return an error without any meaningful information.

Key Assertions:

- The returned warning object has the correct `code` attribute set to `WarningCode.W001_NO_COVERAGE`.
- The returned warning object has a non-empty `message` attribute containing 'No .coverage file found'.
- The returned warning object has a non-empty `detail` attribute containing 'test-detail'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Prevents a potential bug where incorrect warning code values are used, potentially causing unexpected behavior or errors in the application.

Key Assertions:

- {'message': 'assert WarningCode.W001_NO_COVERAGE.value == "W001"', 'description': 'Correctly asserts that WarningCode.W001_NO_COVERAGE has value "W001"'}
- {'message': 'assert WarningCode.W101_LLM_ENABLED.value == "W101"', 'description': 'Correctly asserts that WarningCode.W101_LLM_ENABLED has value "W101"'}
- {'message': 'assert WarningCode.W201_OUTPUT_PATH_INVALID.value == "W201"', 'description': 'Correctly asserts that WarningCode.W201_OUTPUT_PATH_INVALID has value "W201"'}
- {'message': 'assert WarningCode.W301_INVALID_CONFIG.value == "W301"', 'description': 'Correctly asserts that WarningCode.W301_INVALID_CONFIG has value "W301"'}
- {'message': 'assert WarningCode.W401_AGGREGATE_DIR_MISSING.value == "W401"', 'description': 'Correctly asserts that WarningCode.W401_AGGREGATE_DIR_MISSING has value "W401"'}

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 Warning class to ensure it correctly converts warnings into dictionaries.

Why Needed: This test prevents a potential bug where the `Warning.to_dict()` method does not convert warnings with detailed messages into dictionaries as expected.

Key Assertions:

- The warning's code should be 'W001' and its message should be 'No coverage'.
- The warning's detail should be 'some/path'.
- A warning without a detailed message should have its code as 'W101' and its message as 'LLM 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/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 that a warning with known code is created.

Why Needed: Prevents regression in case of unknown WarningCode.LLM_ENABLED.

Key Assertions:

- assert w.code == WarningCode.W101_LLM_ENABLED
- assert w.message == WARNING_MESSAGES[WarningCode.W101_LLM_ENABLED]
- assert w.detail 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/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 Make Warning: Unknown Code

Why Needed: Prevents a potential bug where the fallback message is not used for unknown code.

Key Assertions:

- The function `make_warning` should raise an exception when given an unknown warning code.
- The function `make_warning` should use the fallback message 'Unknown warning.' for unknown codes.
- The value of `WARNING_MESSAGES[missing_code]` should be set to `old_message` before restoring it after the try-except block.
- The value of `WARNING_MESSAGES[missing_code]` should not be changed back to its original value after the try-except block.

COVERAGE

src/pytest_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 created when an invalid configuration detail is provided.

Key Assertions:

- The function `make_warning` returns an instance of `Warning` with the specified `code` and `detail`.
- The `detail` attribute of the returned `Warning` object matches the expected value 'Bad value'.
- The `code` attribute of the returned `Warning` object matches the expected value WarningCode.W301_INVALID_CONFIG.

COVERAGE

src/pytest_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: Verify that enum values are correctly initialized as strings.

Why Needed: This test prevents the `WarningCodes` class from being used with non-string values.

Key Assertions:

- assert isinstance(code.value, str)
- assert code.value.startswith('W')
- code.value should be a string (not an integer or other type)
- code.value should start with 'W' (as defined in the WarningCode enum)
- WarningCodes class should not be used with non-string values

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the `to_dict()` method of a Warning object returns a dictionary with the correct keys and values when no detail is included.

Why Needed: This test prevents a potential issue where the warning data is not properly serialized to JSON without including detailed information about the code coverage.

Key Assertions:

- The `to_dict()` method of the Warning object returns a dictionary with the correct keys: `code` and `message`.
- The value of the `code` key in the returned dictionary matches the expected string 'W001'.
- The value of the `message` key in the returned dictionary matches the expected string 'No coverage'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/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 that Warning to dictionary with detail is correctly serialized.

Why Needed: This test prevents a warning about the serialization of warnings to dictionaries.

Key Assertions:

- The 'code' key should have the value 'W001'.
- The 'message' key should have the value 'No coverage'.
- The 'detail' key should have the value 'Check setup'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	6 lines (ranges: 70-72, 74-76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 .txt and .pyc files as Python files.

Key Assertions:

- The function should return `False` when given a file name without a `.py` extension.
- The function should return `False` when given a file name with a `.pyc` extension (Python compiled code).
- The function should not incorrectly identify files like `foo/bar.txt` as Python files.

COVERAGE

src/pytest_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 returns True for a Python file.

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

Key Assertions:

- The function should correctly identify `.py` files and return `True`.
- The function should not incorrectly identify other types of files (e.g. `txt`, `jpg`) as Python files.
- The function should handle file paths with spaces or special characters correctly.
- The function should raise an error for non-existent file paths.
- The function should ignore case when comparing file names to `.py` extension.
- The function should not perform any additional checks beyond just the file extension.
- The function should be able to handle files in different directories (e.g. `/home/user/foo/bar.py`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 the `make_relative` function correctly makes a path relative to the test directory.

Why Needed: This test prevents regression where the `make_relative` function fails to make paths relative to the test directory.

Key Assertions:

- The file is created with the correct relative path.
- The parent directory of the file is created if it does not exist.
- The file is successfully touched after being made relative.
- The `make_relative` function returns the expected output for the given input.
- The test directory is used as the base directory for making paths relative.
- The `tmp_path` object is used to create and manage temporary directories during testing.

COVERAGE

src/pytest_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 with no base.

Why Needed: Prevents a potential issue where an incorrect or malformed relative path is returned.

Key Assertions:

- The resulting path should be in the format of a valid file system path (e.g., 'foo/bar')
- The path should not contain any leading directory separators (e.g., '../', './', etc.)
- Any trailing directory separators should be removed (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/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 test verifies that the `normalize_path` function correctly handles already-normalized paths.

Why Needed: This test prevents a potential bug where the `normalize_path` function would incorrectly handle normalized paths, leading to unexpected behavior or errors.

Key Assertions:

- The path should be returned unchanged if it is already normalized.
- The path should not be modified by the `normalize_path` function even if it is already normalized.
- If an empty string is passed as input, the function should return an empty string.
- If a non-string input is passed to the function, it should raise a TypeError.
- The function should handle paths with multiple separators correctly (e.g., `/foo/bar` and `foo\bar`).
- The function should not add any additional separators to normalized paths (e.g., `foo//bar` becomes `foo/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: Tests the functionality of the `normalize_path` function when it encounters forward slashes in a path.

Why Needed: This test prevents regressions where the function does not correctly handle paths with forward slashes.

Key Assertions:

- The function should convert '\bar' to '/bar'.
- The function should return 'foo/bar' after processing the input path.
- The function should handle cases where the input path starts or ends with a backslash.
- The function should correctly handle paths that contain multiple consecutive forward slashes.
- The function should not throw any exceptions when encountering invalid characters in the input path.
- The function should preserve the original directory separator used in the input path (e.g., Windows-style vs. Unix-style).
- The function should be able to handle paths with a mix of forward and backslashes.

COVERAGE

src/pytest_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: Tests that the `normalize_path` function strips trailing slashes from file paths.

Why Needed: This test prevents a potential bug where a path with a trailing slash is not stripped correctly.

Key Assertions:

- The input path should be split into components without any trailing slashes.
- The resulting normalized path should have the same components as the original input path.
- No leading or trailing slashes should be present in the normalized path.
- Any directory separator (e.g., `/`) should be replaced with a single `/`.
- The function should not modify the original input path if it already had no trailing slash.
- The function should handle paths with multiple consecutive slashes correctly.
- The function should handle paths with leading or trailing slashes correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: The test verifies that the `should_skip_path` function correctly skips paths matching custom patterns.

Why Needed: This test prevents a potential bug where the `should_skip_path` function does not properly handle custom exclusion patterns.

Key Assertions:

- paths matching custom patterns are excluded from being skipped.
- custom patterns are matched against the provided exclude_patterns list.
- the exclude_patterns list is correctly updated with the custom pattern.
- the `should_skip_path` function returns True for paths that match the custom pattern.
- the `should_skip_path` function returns False for paths that do not match any of the excluded 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/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: tests/test_fs.py::TestShouldSkipPath::test_normal_path

Why Needed: The current implementation of `should_skip_path` may incorrectly skip normal paths, preventing the test from passing.

Key Assertions:

- assert should_skip_path('src/module.py') is False
- assert not should_skip_path('tests/test_fs.py::TestShouldSkipPath::test_normal_path') is True

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: The test verifies that the `should_skip_path` function correctly identifies `.git` directories.

Why Needed: This test prevents a potential issue where the function incorrectly skips non-existent or non-`.git` directories.

Key Assertions:

- assert should_skip_path('.git/objects/foo') is True
- assert should_skip_path('non-existent/.git') 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/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 it skips the __pycache__ directory for files with a '.pyc' extension.

Why Needed: This test prevents a potential issue where the test is incorrectly skipping certain .pyc files due to a misunderstanding of the __pycache__ directory.

Key Assertions:

- assert should_skip_path('foo/__pycache__/bar.pyc') == True
- assert should_skip_path('foo/__pycache__/baz.pyo') == False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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: tests/test_fs.py::TestShouldSkipPath::test_skips_venv verifies that the `should_skip_path` function correctly identifies venv directories.

Why Needed: This test prevents a regression where the function incorrectly considers venv directories as Python site packages.

Key Assertions:

- The function should return True for venv directories (e.g., 'venv/lib/python/site.py')
- The function should return True for .venv directories (e.g., '.venv/lib/python/site.py')
- The function should not consider other Python site packages as venv directories
- The function should raise an exception when given a non-venv directory (e.g., 'non_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 of request times and token usage prevents the accumulation of these metrics after a past request.

Why Needed: This test prevents regression in the GeminiRateLimiter class when it prunes request times and token usage, ensuring that they do not accumulate over time.

Key Assertions:

- The length of _request_times should be zero after pruning.
- The length of _token_usage should be zero 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: Test that the rate limiter prevents requests from exceeding the specified limit for a specific time period.

Why Needed: This test prevents a potential bug where the rate limiter allows too many requests in a short period, potentially leading to performance issues or errors.

Key Assertions:

- The `next_available_in` method should return a wait time greater than 0 seconds.
- The `wait` value should be less than or equal to 60.0 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	26 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95, 97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: This test prevents a potential bug where the rate limiter does not prevent token updates from being recorded before the next available time.

Key Assertions:

- The `next_available_in` method returns a positive value (greater than 0) after waiting for the specified amount of time.
- The `_token_usage` list has two elements when tokens are first updated.
- Tokens are not recorded until the next available time, preventing token updates from being overwritten.

COVERAGE

src/pytest_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: Test that the wait_for_slot method sleeps when a request is recorded but not yet processed.

Why Needed: This test prevents a potential race condition where a request is recorded before it's actually processed, potentially causing the limiter to fail or behave unexpectedly.

Key Assertions:

- limiter.record_request() should be called
- limiter.wait_for_slot(1) should call mock_sleep
- mock_sleep should have been called with argument 1
- the method should not return immediately after recording the 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	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: Test that the rate limiter records zero tokens when no tokens are available.

Why Needed: This test prevents a potential regression where the rate limiter does not record tokens for 0 tokens, potentially leading to incorrect usage statistics.

Key Assertions:

- The `_token_usage` list should be empty after calling `record_tokens(0)`.
- The number of elements in `_token_usage` should be 0.
- The length of `_token_usage` should match the expected value (0) before any tokens are recorded.

COVERAGE

src/pytest_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 a `GeminiRateLimitExceeded` exception when exceeding daily requests.

Why Needed: Prevents regression in rate limiting logic where daily limit is exceeded.

Key Assertions:

- The function `_GeminiRateLimiter.wait_for_slot(10)` should raise an error with the message 'requests_per_day':
- The `GeminiRateLimitConfig` instance has a valid `requests_per_day` value of 1.
- The `record_request()` method is called before attempting to wait for a slot.
- The `wait_for_slot(10)` method raises an error with the specified message.
- _GeminiRateLimiter.wait_for_slot(10) should raise a `GeminiRateLimitExceeded` exception
- The limit of 1 requests per day is not exceeded by calling `_GeminiRateLimiter.wait_for_slot(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	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 TPM fallback waits for enough time when tokens are used beyond the rate limit.

Why Needed: This test prevents a potential regression where the TPM limiter does not wait long enough for the tokens to be available before falling back to the fallback mechanism.

Key Assertions:

- The `wait` variable is greater than 0 after filling up TPM (line 116).
- Tokens are used beyond the rate limit (`tokens_used + request_tokens > limit`) and token usage is not empty (line 116).

COVERAGE

src/pytest_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 600ms 6

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the RPM rate limit cooldown might not be properly set on the first call to _call_gemini.

Key Assertions:

- The 'models/gemini-pro' key should be present in the '_cooldowns' dictionary.
- 'models/gemini-pro' should have a value greater than 1000.0.
- The cooldown time for 'models/gemini-pro' should be at least 1 second.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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 correctly annotates a rate limit retry scenario.

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

Key Assertions:

- The annotation should have the correct scenario.
- The mock_post call count should be equal to 2.
- The annotation should not throw an error when retrying a failed request.
- The provider's internal state should reflect that it has been retried.
- The annotation should contain the expected model name and supported generation methods.
- The provider should correctly handle rate limit retries by waiting for the specified time before retrying the request.
- The mock_get call should return a 429 status code with the Retry-After header set to 0.1 seconds.
- The mock_post call should be one of the two successful responses (200 or 429).

COVERAGE

src/pytest_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 4ms 4

AI ASSESSMENT

Scenario: Verify that the _annotate_internal method returns a correct LlmAnnotation object with the expected scenario.

Why Needed: This test prevents regression where the _parse_response method expects an incorrect format of response from _call_gemini.

Key Assertions:

- The annotation returned by _annotate_internal has the correct scenario set to 'Success Scenario'.
- The annotation does not have any error.
- The annotation is of type LlmAnnotation.
- The annotation's source attribute matches the expected value 'source'.
- The annotation's error attribute is None.
- The annotation's text attribute contains the expected part 'Success 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	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 3ms 5

AI ASSESSMENT

Scenario: Verify that the `GeminiProvider` class correctly checks availability based on environment settings.

Why Needed: This test prevents a potential bug where the `GeminiProvider` class does not correctly check for availability when the GEMINI_API_TOKEN environment variable is set.

Key Assertions:

- The `_check_availability()` method of the `GeminiProvider` class should return False if the `GEMINI_API_TOKEN` environment variable is not set.
- The `_check_availability()` method of the `GeminiProvider` class should return True if the `GEMINI_API_TOKEN` environment variable is set and a valid token is provided.
- The `Config` instance passed to the `GeminiProvider` constructor should have a 'provider' key with value 'gemini'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 ensures that the rate limiter does not allow more than one request to be processed within a single day, preventing potential abuse or denial-of-service attacks.

Key Assertions:

- limiter.next_available_in(100) should return None when there are no available slots in the rate limit.
- limiter.next_available_in(101) should raise an exception when the daily limit is exceeded.

COVERAGE

src/pytest_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: Verify that the rate limiter does not allow more than 2 requests per minute for a single provider.

Why Needed: This test prevents a potential performance regression where the rate limiter allows too many requests to be processed in a short time, potentially leading to slow or unresponsive behavior.

Key Assertions:

- The next_available_in method returns 0.0 after the first two requests.
- The next_available_in method returns 0.0 after the third request has been recorded but before it is allowed to process.
- The wait value is greater than 0 and less than or equal to 60.0 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	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 configuration providers produce different hashes.

Why Needed: Prevents regression in case of provider changes, ensuring consistent hash generation.

Key Assertions:

- A config with provider 'none' should have a hash different from one with provider 'ollama'.
- The hash for a config with provider 'ollama' should be different from the one with provider 'none'.
- Different providers (none vs. ollama) should produce distinct hashes.
- A config with provider 'ollama' should not match the hash of a config with provider 'none'.
- The hash for a config with provider 'ollama' should be different from one without it.
- Config with provider 'ollama' and no other providers should have a unique hash.
- Different configuration providers (none vs. ollama) should result in distinct hashes.

COVERAGE

src/pytest_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: Verify that the computed hash is of length 16.

Why Needed: This test 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 due to padding.
- No padding is applied if the input configuration is valid.
- The hash is generated using the correct algorithm and data.
- The hash calculation does not produce a hash that can be hashed again without modification.
- The hash is unique for each valid configuration.
- A hash of length 15 or less is returned when the input configuration has fewer than 16 bytes.

COVERAGE

src/pytest_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.

Why Needed: Prevents a potential bug where the computed hash does not match the actual content hash due to differences in data representation or encoding.

Key Assertions:

- The computed SHA-256 hash of the file should be equal to the actual content hash.
- The computed SHA-256 hash of the file should be identical to the content hash.
- The computed SHA-256 hash of the file should match the expected value based on its content.
- The computed SHA-256 hash of the file should not change unexpectedly when the same data is used.
- The computed SHA-256 hash of the file should remain unchanged even if the data is modified or corrupted.
- The computed SHA-256 hash of the file should be consistent across different platforms and environments.

COVERAGE

src/pytest_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: Prevents potential issues with incorrect or corrupted file contents being hashed.

Key Assertions:

- The length of the computed hash should be exactly 64 bytes.
- The computed hash should not contain any zeros (indicating an invalid input).
- The computed hash should match the expected SHA-256 hash value for the given file contents.

COVERAGE

src/pytest_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 bug where the same input produces the same signature for different keys, potentially leading to security vulnerabilities or unexpected behavior.

Key Assertions:

- The computed HMAC signature is different from the expected signature when using different keys.
- The computed HMAC signature does not match the expected signature when using the same key but a different input.
- The computed HMAC signature is different from the expected signature when using different key sizes (e.g., 128-bit vs. 256-bit).
- The computed HMAC signature matches the expected signature when using the same key and a different input size (e.g., 32-byte vs. 64-byte)
- The computed HMAC signature does not match the expected signature when using different key sizes for the same input.
- The computed HMAC signature is different from the expected signature when using different key formats (e.g., base64-encoded vs. binary-encoded).
- The computed HMAC signature matches the expected signature when using the same key and a different encoding 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/hashing.py	1 lines (ranges: 61)

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

AI ASSESSMENT

Scenario: Verify the length of the generated HMAC signature.

Why Needed: Prevents a potential issue where an attacker could manipulate the input data to produce an invalid or shorter signature.

Key Assertions:

- The length of the generated HMAC signature is 64 bytes.
- The generated HMAC signature contains all required padding characters (OAEP padding).
- The generated HMAC signature includes the secret key as a prefix, ensuring it can be verified 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/hashing.py	1 lines (ranges: 61)

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

AI ASSESSMENT

Scenario: Test that the SHA-256 hash of the same content is consistent.

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

Key Assertions:

- The two computed hashes should be equal.
- The first and second hashes should have the same value.
- The hash of the same content should not change even if the input is modified.
- The hash of a copy of the input should also be equal to the original hash.
- If the input is modified, the computed hash should remain unchanged.
- If the input is copied, the computed hash should also remain unchanged.
- The hash of the same content produced by different inputs 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	1 lines (ranges: 32)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the hash length is not correctly calculated due to incorrect input size or data type.

Key Assertions:

- The length of the computed hash should be exactly 64 bytes (64 * 16 = 1024).
- The hash value should contain exactly 64 hexadecimal characters (0-9, A-F, a-f).
- No leading zeros are allowed in the hash value.
- No trailing zeros are allowed in the hash value.
- The hash value is not empty.
- All hexadecimal digits are present and correctly formatted.
- There are no duplicate hexadecimal values in the hash 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/hashing.py	1 lines (ranges: 32)

PASSED `tests/test_hashing.py::TestGetDependencySnapshot::test_includes_pytest` 80ms 3

AI ASSESSMENT

Scenario: Verifies that the `get_dependency_snapshot()` function includes the 'pytest' package in its output.

Why Needed: This test prevents a regression where the 'pytest' package is not included in the dependency snapshot due to an incorrect implementation of the `get_dependency_snapshot()` function.

Key Assertions:

- The function should return a string containing the 'pytest' package name.
- The function should include the 'pytest' package name in its output, regardless of any other dependencies it may have.
- Any additional dependencies included by the 'pytest' package should be reported as well.
- If the 'pytest' package is not installed or available, an error message indicating this should be displayed.
- The function should handle cases where the 'pytest' package has a specific version requirement.
- The output of the `get_dependency_snapshot()` function should always include the 'pytest' package name.
- Any other dependencies included by the 'pytest' package should be reported as well, even if they are not part of the main dependency.
- If the 'pytest' package is not installed or available, an error message indicating this should be displayed.

COVERAGE

src/pytest_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 82ms 3

AI ASSESSMENT

Scenario: The test verifies that the `get_dependency_snapshot` function returns a dictionary.

Why Needed: This test prevents a potential bug where the function might not return a dictionary or may return unexpected data.

Key Assertions:

- snapshot is an instance of dict
- snapshot has no attributes other than __dict__
- snapshot does not contain any non-numeric keys
- snapshot contains only numeric keys
- snapshot contains only string keys
- snapshot is empty
- snapshot contains a single key-value pair
- snapshot contains multiple key-value pairs

COVERAGE

src/pytest_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 potential bug where the HMAC key is not loaded correctly if the file does not exist or cannot be read.

Key Assertions:

- The output of the `load_hmac_key(config)` function should match the expected value (`b'my-secret-key'`),
- The `config.hmac_key_file` attribute should point to the correct file (`tmp_path / 'hmac.key'`).
- The HMAC key loaded from the file should be identical to the expected value (`b'my-secret-key'`).
- The `load_hmac_key(config)` function should raise an error if the file does not exist or cannot be read.
- The `load_hmac_key(config)` function should handle cases where the file is empty (e.g., due to a corrupted 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	5 lines (ranges: 73, 76-77, 80-81)

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

AI ASSESSMENT

Scenario: The test verifies that the `load_hmac_key` function returns `None` when a non-existent key file is provided.

Why Needed: This test prevents a potential bug where the function fails to return an error message or handle the case of a missing key file.

Key Assertions:

- The function should raise a `ValueError` exception with a meaningful error message when the key file does not exist.
- The function should check if the provided path is absolute before attempting to create it.
- The function should return an empty string or a default value (e.g., None) instead of raising an exception.
- The function's behavior should be consistent across different Python versions and platforms.
- The test should cover both Windows and Unix-like systems when verifying the key file path.
- The test should also verify that the error message is not too trivial or misleading.

COVERAGE

src/pytest_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: Test that the `load_hmac_key` function returns `None` when no key file is specified.

Why Needed: Prevents a potential bug where the test fails with an error message indicating a missing key file, instead of returning a meaningful result.

Key Assertions:

- The `load_hmac_key` function should return `None` if no key file is provided.
- The `load_hmac_key` function should not raise any exceptions or errors when no key file is specified.
- The test should verify that the returned value matches the expected result (`None` 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/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: Verify that aggregation defaults are set correctly.

Why Needed: This test prevents a potential bug where the default aggregation policy is not set to 'latest'.

Key Assertions:

- config.aggregate_dir is None (expected)
- config.aggregate_policy == 'latest' (expected)
- config.aggregate_include_history is False (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	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 potential bug where the default capture failed output setting is not correctly configured, causing unexpected behavior in the integration gate.

Key Assertions:

- config.capture_failed_output should be False
- config.capture_failed_output is not True
- config.capture_failed_output is 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_context_mode_default_minimal` 1ms 3

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the context mode is not set to 'minimal' by default.

Key Assertions:

- config.llm_context_mode == 'minimal'
- assert isinstance(config.llm_context_mode, str)
- assert config.llm_context_mode in ['minimal', 'none']
- assert config.context_mode != 'none'
- assert config.context_mode != 'default'
- assert config.context_mode not in ['none', 'default']
- assert get_default_config().llm_context_mode == 'minimal'

COVERAGE

src/pytest_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 regression where LLM is enabled by default due to a bug or change in the configuration settings.

Key Assertions:

- The `is_llm_enabled()` method returns False for the default configuration.
- The `get_default_config()` function returns an object with an `llm_enabled` attribute set to False.
- The `llm_enabled` attribute is not initialized with a value of True by default in the default configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: Verify that the `TestConfigDefaults` class sets `omit_tests_from_coverage` to `True` when `default_omit_tests` is set to `True`.

Why Needed: This test prevents a regression where the default coverage omission setting does not apply correctly for tests with `default_omit_tests=True`.

Key Assertions:

- The `config.omit_tests_from_coverage` attribute of the `TestConfigDefaults` instance is set to `True`.
- The `get_default_config()` method returns an instance of `TestConfigDefaults` with `omit_tests_from_coverage=True`.
- The `assert` statement checks that `config.omit_tests_from_coverage` is equal to `True`.
- The `config` attribute is accessed and its `omit_tests_from_coverage` property is checked.
- The `get_default_config()` method is called and the resulting instance of `TestConfigDefaults` is stored in the `config` variable.
- The `assert` statement checks that the `config` object has an `omit_tests_from_coverage` attribute with a value of `True`.
- The `default_omit_tests` property of the `TestConfigDefaults` instance is set to `True` and checked for consistency with the expected 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' in case of privacy requirements.

Key Assertions:

- The function get_default_config() returns a config object with a provider attribute set to 'none'.
- The assert statement checks if the config.provider equals 'none'.
- If the provider was set to another value, this test would fail and raise an AssertionError.
- This ensures that the default provider is correctly set when privacy requirements are not met.
- The function get_default_config() returns a config object with a provider attribute set to 'None' (not 'none').
- If the provider was set to another value, this test would fail and raise an AssertionError.
- This ensures that the default provider is correctly set when privacy requirements are not met.
- The function get_default_config() returns a config object with a provider attribute set to None.
- If the provider was set to another value, this test would fail and raise an AssertionError.

COVERAGE

src/pytest_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 secret files might be accidentally included in the LLM context.

Key Assertions:

- The 'secret' keyword is present in the excludes list for any glob pattern.
- The '.env' file is also present in the excludes list for any glob pattern.
- Any other secret files or directories that should not be included in the LLM context are excluded.
- If a non-secret file is accidentally added to the excludes list, it will still be excluded from the LLM context.
- The excludes list only includes specific keywords and patterns that match known secret files or directories.
- Any other excluded items are ignored by the test.
- The test ensures that all secret files are properly excluded from 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` 6ms 5

AI ASSESSMENT

Scenario: The test verifies that the reports are deterministic (sorted by nodeid) and reports are generated correctly.

Why Needed: This test prevents a regression where the order of node IDs in the report is not guaranteed to be sorted.

Key Assertions:

- # Tests are added to the report correctly
- # The tests are ordered in ascending order (node ID)
- # Node IDs are unique and do not repeat
- # Tests with different node IDs have different node IDs
- # Tests with identical node IDs have the same node ID
- # Tests with duplicate node IDs have a unique node ID for each test
- # The report is sorted in ascending order (node ID)
- # The report does not contain any duplicate node IDs

COVERAGE

src/pytest_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 the pipeline is expected to produce an error when given no input.

Key Assertions:

- The 'summary.total' field of the report should be set to 0.
- Data from the report.json file should contain a 'summary' section with a 'total' key equal to 0.
- The test suite should not produce any errors when given no input data.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 31ms 5

AI ASSESSMENT

Scenario: Test the full pipeline's ability to generate an HTML report.

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

Key Assertions:

- The script checks if the HTML file exists at `tmp_path / 'report.html'`.
- The script reads the contents of the HTML file using `html_path.read_text()` and checks if it contains the string '
- The script asserts that the presence of 'tests/test.py::test_pass' in the HTML content indicates a successful test pass.

COVERAGE

src/pytest_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 the full pipeline generates a valid JSON report.

Why Needed: This test prevents regression in the JSON report generation process, ensuring consistency with previous test results.

Key Assertions:

- The 'schema_version' key is present and correctly set to SCHEMA_VERSION.
- The 'summary' section contains the expected number of tests (3), passed (1), failed (1), and skipped (1).
- All test nodes are included in the report, with no missing or duplicate results.

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 has required fields.

Why Needed: This test prevents a potential bug where the report root is missing required fields, which could lead to incorrect or incomplete reports.

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' list should contain at least one element.
- All required fields ('schema_version', 'run_meta', and 'summary') 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: Verify that the `RunMeta` object has 'is_aggregated' and 'run_count' fields.

Why Needed: Prevents regression where a schema with aggregation fields does not have these required fields.

Key Assertions:

- The 'is_aggregated' field is present in the data.
- The 'run_count' field is present in the data.
- The presence of 'is_aggregated' and 'run_count' fields ensures that RunMeta has aggregation capabilities.
- Without these fields, a schema with aggregation would not be compatible with other schemas.

COVERAGE

`src/pytest_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: Verify that RunMeta contains status fields.

Why Needed: Prevents regression where RunMeta is expected to be missing certain status fields.

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: Verify 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 compatibility issues with other components.

Key Assertions:

- SCHEMA_VERSION is defined and contains at least one dot (.), indicating it's a valid semver-like string.
- The presence of a dot in SCHEMA_VERSION indicates that the schema version is compatible with semantic versions.
- A valid semver-like string is expected for the schema version, ensuring compatibility with other components.

COVERAGE

`src/pytest_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` test verifies that the `TestCaseResult` object has the required fields.

Why Needed: This test prevents a potential bug where the `TestCaseResult` object is missing required fields, which could lead to incorrect analysis or reporting.

Key Assertions:

- The 'nodeid' field should be present in the `data` dictionary.
- The 'outcome' field should be present in the `data` dictionary.
- The 'duration' field 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 an instance of `GeminiProvider` when the `provider` parameter is set to `'gemini'`.

Why Needed: This test prevents a potential bug where the `get_provider` function does not return an instance of `GeminiProvider` when the `provider` parameter is set to `'gemini'`, potentially causing issues with downstream code that relies on this behavior.

Key Assertions:

- The `get_provider` function should return an instance of `GeminiProvider`.
- The `get_provider` function should return an instance of `GeminiProvider` when the `provider` parameter is set to `'gemini'`.
- The `get_provider` function should not raise any exceptions when the `provider` parameter is set to `'gemini'`.

COVERAGE

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

Why Needed: This test prevents a potential bug where the correct provider is not returned for a given model.

Key Assertions:

- provider.__class__.__name__ == 'LiteLLMProvider'
- get_provider(config).model == 'gpt-3.5-turbo'
- get_provider(config).provider == 'litellm'

COVERAGE

src/pytest_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 returns NoopProvider when provider is 'none'

Why Needed: This test prevents a potential regression where the LLM's default provider is set to 'none' and it doesn't return a NoopProvider.

Key Assertions:

- config.provider should be 'none'
- provider should be an instance of NoopProvider
- assert isinstance(provider, NoopProvider) 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/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 when a specific provider ('ollama') is used, the returned OllamaProvider object matches its expected type.

Why Needed: This test prevents a potential bug where using 'ollama' as a provider might cause an import error if the httpx library is not properly installed or imported.

Key Assertions:

- The provider should be of type OllamaProvider.
- The provider's class name should match 'OllamaProvider'.
- The provider object should have the correct class name when instantiated.
- The provider object should be an instance of OllamaProvider, not another type.
- The provider object should have a __class__ attribute that is set to OllamaProvider.
- The provider's __name__ attribute should match 'OllamaProvider'.
- The provider's class name should exactly 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.

Why Needed: To prevent unexpected behavior when using an unknown provider.

Key Assertions:

- The function `get_provider` should raise a `ValueError` with the message 'unknown' when called with an unknown provider.
- The error message should include the string 'unknown'.
- The `str(e).lower()` method should be used to check if the error message contains 'unknown'.

COVERAGE

src/pytest_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 the NoopProvider class implements all required LlmProvider interface methods.

Why Needed: This test prevents a potential regression where the NoopProvider class does not implement all required methods of the LlmProvider interface.

Key Assertions:

- The NoopProvider should have the annotate method.
- The NoopProvider should have the is_available method.
- The NoopProvider should have the get_model_name method.
- The NoopProvider should have the 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 NoopProvider returns an empty annotation when no annotations are provided.

Why Needed: This test prevents a regression where the NoopProvider does not return any annotation when no annotations are provided.

Key Assertions:

- annotation is of type LlmAnnotation
- annotation scenario is an empty string
- annotation why_needed is an empty string
- annotation key_assertions is an empty list

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the `get_model_name` method returns an empty string when the input configuration is empty.

Why Needed: This test prevents a potential bug where the `get_model_name` method throws an exception or raises an error when given an invalid or empty configuration.

Key Assertions:

- The `get_model_name()` method of the `NoopProvider` class should return an empty string.
- The `get_model_name()` method of the `NoopProvider` class should not throw any exceptions or raise any errors for an empty input configuration.
- The `get_model_name()` method of the `NoopProvider` class should behave as expected when given a valid, non-empty configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/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: Verify that the NoopProvider instance is available.

Why Needed: Prevent regression in case the provider is not available due to a bug or configuration issue.

Key Assertions:

- provider.is_available() should return True
- provider.is_available() should be checked before calling any methods on it
- NoopProvider instance should always have an 'is_available' method that returns True

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 107, 110-111)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 58)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function `test_annotate_tests_emits_summary` should print 'Annotated 1 test(s) via litellm' to the console.
- The function `test_annotate_tests_emits_summary` should capture and return the output of the annotation process.
- The function `test_annotate_tests_emits_summary` should assert that the captured output contains the expected string.
- The function `test_annotate_tests_emits_summary` should not fail when run without any errors.

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 sent correctly when annotating tests.

Why Needed: This test prevents regression where the progress report may not be sent or may be incomplete.

Key Assertions:

- The progress message should contain the correct scenario and test name.
- The progress message should include the correct provider.
- The progress message should have the expected number of messages.
- Each message in the progress list should match the expected format.
- The test case's outcome should be passed as expected.
- The LLM annotation should be completed correctly for all tests.
- The provider should be set to a valid instance of FakeProvider.
- The progress callback should append messages to the correct list.

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

Why Needed: To prevent regression in the LLM annotator, this test verifies that it respects the opt-out setting and limits the number of tests to 1.

Key Assertions:

- The LLM annotator should not call `get_provider` with an invalid configuration.
- The LLM annotator should only annotate tests where `llm_opt_out` is set to True.
- The LLM annotator should not annotate more than one test per nodeid.
- The LLM annotator should not annotate a test that has already been annotated.
- The LLM annotator should respect the limit on the number of tests.
- The LLM annotator should not call `get_provider` with an invalid configuration after limiting the number of tests to 1.
- The LLM annotator should only be able to annotate one test per nodeid.
- The LLM annotator should not be able to annotate more than one test per nodeid if opt-out is set to True.

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: The test verifies that the LLM annotator respects the requests-per-minute rate limit.

Why Needed: This test prevents a potential bug where the LLM annotator exceeds the allowed requests per minute, potentially causing performance issues or errors.

Key Assertions:

- The provider function should be called with the correct list of node IDs.
- The sleep_calls variable should contain the expected values for the monotonic and sleep functions.
- The calls to the provider function should match the provided tests.
- The sleep_calls variable should contain a single value (2.0) after the test has completed.
- The key_assertions list should not be empty.

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 fails when unavailable provider is specified.

Why Needed: To prevent regression where annotation fails due to an unavailable provider.

Key Assertions:

- The function `get_provider` from `pytest_llm_report.llm.annotator` should be set to the provided UnavailableProvider instance.
- The annotation process should fail when trying to annotate tests with an unavailable provider.
- A message indicating that the provider is not available should be printed to the console.
- The function `is_available` of the UnavailableProvider class should return False.
- The annotation process should skip annotation if the provider is unavailable.

COVERAGE

src/pytest_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 the annotation function to ensure it uses a cache.

Why Needed: This test prevents regression where the annotations are not cached between runs, potentially leading to inconsistencies in the results.

Key Assertions:

- The `get_provider` method of `pytest_llm_report.llm.annotator` is called with the correct configuration.
- The annotation function does not call `pytest_llm_report.llm.annotator.get_provider` when it should not.
- The annotation function sets `llm_annotation` to a valid object after calling `get_provider`.
- The annotation function checks if `llm_annotation` is `None` before setting it, ensuring it's only set when necessary.
- The annotation function does not call `pytest_llm_report.llm.annotator.get_provider` again after setting `llm_annotation` to a valid object.

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: The test verifies that the `test_required_fields` function checks for the presence of 'scenario' and 'why_needed' in the annotation JSON schema.

Why Needed: This test prevents a potential bug where the LLM contract's schema does not require these two fields, potentially leading to incorrect validation or errors during testing.

Key Assertions:

- The function `test_required_fields` is called with an assertion that checks for the presence of 'scenario' and 'why_needed' in the annotation JSON schema.
- The test asserts that both 'scenario' and 'why_needed' are present in the required fields array.
- The function uses the `get()` method to retrieve the required fields from the annotation JSON schema, which returns an empty list if no fields are specified.
- If the LLM contract's schema does not require these two fields, the test will fail with a meaningful error message indicating the missing requirements.
- The test also verifies that both 'scenario' and 'why_needed' are present in the required fields array to ensure consistency across different scenarios.
- The function uses an assert statement to verify the presence of 'scenario' and 'why_needed' in the required fields array, which will raise an AssertionError if they are not found.

COVERAGE

src/pytest_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: The test verifies that the AnnotationSchema can correctly parse a dictionary containing required fields.

Why Needed: This test prevents potential authentication bypass vulnerabilities by ensuring that the AnnotationSchema is able to extract necessary information from a valid user login scenario.

Key Assertions:

- checks password
- checks username

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the AnnotationSchema does not validate or handle empty inputs.

Key Assertions:

- schema.scenario = "" (empty string)
- schema.why_needed = "" (no validation for empty inputs)
- assert schema.scenario == "" (checks if the expected scenario is empty)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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 correctly handles a partial input scenario.

Why Needed: This test prevents bugs or regressions where the AnnotationSchema does not handle partial inputs correctly.

Key Assertions:

- assert schema.scenario == 'Partial only'
- assert schema.why_needed == ''
- schema.scenario should be equal to 'Partial only' when input is a partial dictionary
- schema.why_needed should be an empty string when input is a partial dictionary
- The AnnotationSchema should not throw any errors for invalid inputs
- The AnnotationSchema should handle the scenario correctly without throwing 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/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 schema has required fields.

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

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

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

AI ASSESSMENT

Scenario: Test the functionality of AnnotationSchema to serialize to dict.

Why Needed: This test prevents regression by ensuring that AnnotationSchema correctly serializes its internal state to a dictionary.

Key Assertions:

- The 'scenario' key is present in the serialized data and contains the correct value.
- The 'why_needed' key is present in the serialized data and contains the correct value.
- The 'key_assertions' list is present in the serialized data and contains all expected assertions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: Verify that the `test_noop_from_factory` test verifies that a factory configuration with 'none' as the provider returns a NoopProvider.

Why Needed: This test prevents regression where a factory configuration with 'none' as the provider is not correctly returning a NoopProvider.

Key Assertions:

- The `provider` attribute of the returned `NoopProvider` instance should be set to `'none'`.
- The `provider` attribute of the returned `NoopProvider` instance should match the expected value.
- The `provider` attribute of the returned `NoopProvider` instance should not be set to a different provider than '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	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 test verifies that the `NoopProvider` class correctly implements the `LlmProvider` interface.

Why Needed: This test prevents a potential bug where an instance of `NoopProvider` is incorrectly identified as an `LlmProvider` due to incorrect implementation or configuration.

Key Assertions:

- The `provider` variable should be an instance of `LlmProvider`.
- The `provider` variable should have the correct attributes (e.g., `llm_model`, `training_data`) for an `LlmProvider` instance.
- The `provider` variable should not have any additional attributes or methods that are not part of the `LlmProvider` interface.
- The `provider` variable's type should be correctly set to `NoopProvider` based on its implementation.
- Any assertions made about the `provider` variable should only check for the existence and correct types, not its behavior or attributes.
- The test should fail if an instance of `NoopProvider` is incorrectly identified as an `LlmProvider` due to incorrect implementation or configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/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 have any dependencies or side effects.

Why Needed: This test prevents a regression where the NoopProvider returns an incorrect annotation for tests that do not depend on any external resources.

Key Assertions:

- The annotation returned by the NoopProvider should be empty.
- The annotation returned by the NoopProvider should indicate that the test function does not have any dependencies or side effects.
- The annotation returned by the NoopProvider should not contain any information about the test function's execution 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/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 an instance of LlmAnnotation-like object.

Why Needed: This test prevents regression where the annotation result is not properly formatted or does not contain expected attributes.

Key Assertions:

- has attribute 'scenario'
- has attribute 'why_needed'
- has attribute '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_handles_empty_code` 1ms 5

AI ASSESSMENT

Scenario: Provider handles empty code gracefully when given an empty code snippet.

Why Needed: This test prevents a potential bug where the contract does not handle empty code correctly.

Key Assertions:

- The `annotate` method should return a non-None result even if the provided code is empty.
- The `nodeid` and `outcome` parameters are set to valid values in this case.
- The `result` variable is checked for `is not None`, ensuring it has a value.
- The `provider.annotate` method is called with the given test, nodeid, outcome, and an empty code string.
- The `annotate` method should return a result object with a valid status.
- The `status` attribute of the result object is checked to ensure it is not 'failed'.
- If the provider returns a failed result, the test would fail due to the missing key assertion.
- In this case, since the provider does not return a failed result, the test passes without any issues.

COVERAGE

src/pytest_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: Test the provider's handling of None context in the contract.

Why Needed: This test prevents a potential regression where the provider may throw an exception or return an error when given a None context.

Key Assertions:

- The `provider.annotate` method should not raise an exception or return an error when given a `None` context.
- The `provider.annotate` method should return a non-None result object.
- The `test_result` object returned by `provider.annotate` has the correct nodeid and outcome.
- The `provider.annotate` method does not throw any exceptions or raise errors when given a `None` context.
- The `provider.annotate` method returns an error message or exception when given a `None` context, as expected.
- The `provider.annotate` method handles None context correctly in the test environment.

COVERAGE

src/pytest_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 contract functionality when using different models.

Key Assertions:

- The provider has an `annotate` attribute.
- The `annotate` function is callable.
- The `annotate` function is present on the provider object.
- The `annotate` function is not None for all providers.
- The `annotate` function is a method of the provider object, not a property.
- The `annotate` function has the correct signature (no arguments).
- The `annotate` function does not throw an exception when called with no arguments.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 being tested when it handles a context that is too large.

Why Needed: This test prevents a potential bug where the `annotate` method fails due to an excessive amount of data being processed, leading to performance issues or errors.

Key Assertions:

- The `annotate` method should be able to handle contexts without exceeding a certain threshold of data.
- The `annotate` method should not raise any exceptions when handling large contexts.
- The `annotate` method should update the model's state correctly after annotating the context.
- The `annotate` method should not return an error or warning message when handling large contexts.
- The `annotate` method should be able to handle multiple annotations without exceeding the threshold of data.
- The `annotate` method should be able to handle different types of annotations (e.g. text, image, etc.) without issue.
- The `annotate` method should update the model's state correctly after annotating multiple contexts in a row.

COVERAGE

src/pytest_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: Test that LiteLLMProvider annotates a missing dependency correctly.

Why Needed: This test prevents the LiteLLMProvider from incorrectly reporting a missing dependency, which could lead to incorrect annotations and potentially cause issues with the model's behavior.

Key Assertions:

- The annotation error message should include the name of the missing dependency.
- The annotation error message should be in the correct format (e.g., 'litellm not installed. Install with: pip install litellm').
- The annotation error message should indicate that the dependency is required for the model to function 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, 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 the `GeminiProvider` requires an API token and provides a meaningful error message when it's missing.

Why Needed: The current implementation does not prevent the test from passing if the API token is not set, which could lead to unexpected behavior or errors in other parts of the code.

Key Assertions:

- The `annotation.error` attribute should be set to 'GEMINI_API_TOKEN is not set'.
- The error message should indicate that the API token is required for annotation.
- The test case should fail if the API token is missing, and a meaningful error message should be 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	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 by the Gemini provider.

Why Needed: Prevents regressions in token usage tracking.

Key Assertions:

- The 'json' key is present in the captured dictionary.
- The 'status ok' assertion is present in the response metadata.
- The totalTokenCount of the 'usageMetadata' key matches the expected value (123).
- The 'candidates' key contains a list with one element that has a 'text' attribute containing the response data.
- The 'models/gemini-1.5-pro' model is supported and has a 'generateContent' generation method.
- The rate limits for tokens per minute are correctly applied (1000 tokens per minute).
- The limiter's token usage list contains only one element with the expected value (123 tokens).

COVERAGE

src/pytest_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 verifies that the LLM provider annotates retries on rate limits.

Why Needed: This test prevents a potential regression where the LLM provider does not retry when rate limiting occurs.

Key Assertions:

- The LLM provider should retry after rate limiting.
- The retry attempts should be limited to a reasonable number of times.
- The provider should attempt to annotate retries within a short time frame (e.g., 1-2 seconds).
- The annotation should include the reason for the rate limit violation (if any).
- The provider should not retry immediately after the rate limit is lifted.
- The retry attempts should be retried with a different method (e.g., API call) if possible.
- The LLM provider should handle rate limiting correctly and avoid blocking indefinitely.
- The test should fail when rate limiting occurs without any retries.

COVERAGE

src/pytest_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 `annotate` method of the `GeminiProvider` class rotates models on the daily limit when provided with a large number of annotations.

Why Needed: This test prevents regression in the `annotate` method, which could cause it to fail or produce incorrect results for users who have annotated multiple times within a short period.

Key Assertions:

- The `rotate_models_on_daily_limit` method is called with the correct number of annotations.
- The rotated models are not reused across subsequent calls to `annotate`.
- The total number of annotations remains below the daily limit after calling `annotate` multiple times.
- The `rotate_models_on_daily_limit` method does not modify the original model.
- The `annotate` method returns a new instance with the rotated models, rather than modifying the existing one.
- The `rotate_models_on_daily_limit` method is called only once per test run.
- The daily limit is checked and exceeded before calling `rotate_models_on_daily_limit`.
- The `rotate_models_on_daily_limit` method updates the model's metadata 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	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 limits for LLM providers.

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

Key Assertions:

- The `annotate` method should not call `self.daily_limit.skip_daily()` when called within a day.
- The `annotate` method should set `self.daily_limit` to `None` after skipping daily limits.
- The `annotate` method should not raise an exception when called outside of a day.
- The `DailyLimit` class should be implemented correctly to handle daily skips.
- The `DailyLimit` class should have a correct `skip_daily()` method that returns `True` for the provider.
- The `DailyLimit` class should have a correct `is_daily_limit_skipped()` method that returns `False` for the provider.
- The `DailyLimit` class should be able to handle multiple daily limits for different 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	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 the annotate method correctly annotates a successful response from LiteLLM with relevant information.

Why Needed: Prevents regressions caused by incorrect or missing annotations in responses from LiteLLM providers.

Key Assertions:

- The annotation contains the correct scenario and why needed information.
- The annotation includes key assertions that are present in the response payload.
- The confidence level of the annotation is set to 0.8, indicating a high degree of certainty.
- The model used for the annotation is correctly identified from the response payload.
- The messages associated with the annotation contain relevant information about the test case.
- The test function `test_login` is 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 provider's exhausted model should recover after 24 hours of inactivity.

Why Needed: This test prevents a potential regression where the model does not recover from prolonged inactivity.

Key Assertions:

- The `model` attribute is set to `None` after 24 hours.
- The `model` attribute is restored to its original value within 24 hours.
- The `model` attribute's size is less than the initial size after 24 hours.
- The model's training state is updated correctly when it recovers from inactivity.
- The model's predictions are accurate and consistent with the previous predictions after recovery.
- The `model` attribute is not set to `None` or `null` within 24 hours of inactivity.
- The `model` attribute's size increases by a reasonable amount (e.g., 10%) after 24 hours.

COVERAGE

src/pytest_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 no available models are found.

Why Needed: This test prevents a potential bug where the `fetch_available_models` method returns an error when there are no available models, potentially causing unexpected behavior or errors in downstream code.

Key Assertions:

- The `fetch_available_models` method should return an empty list when no available models are found.
- No exception should be raised when calling `fetch_available_models` with a non-empty list of available models.
- The error message returned by the `fetch_available_models` method should indicate that there are no available models.
- The `fetch_available_models` method should return an empty list instead of raising an exception when called with a non-empty list of available models.
- The `fetch_available_models` method should not raise any exceptions when called with a non-empty list of available models.
- The error message returned by the `fetch_available_models` method should be more informative than just 'No available models found'.
- The `fetch_available_models` method should return an empty list instead of raising an exception when called with a non-empty list of available models.

COVERAGE

src/pytest_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 when the LLM provider is not used.

Why Needed: This test prevents a potential issue where the model list is not updated after an interval, potentially leading to incorrect results or errors.

Key Assertions:

- The `model_list` attribute of the `GeminiProvider` instance should be empty before calling `refresh_interval()`
- The `model_list` attribute of the `GeminiProvider` instance should contain a list of models after calling `refresh_interval()`
- The `refresh_interval()` method of the `GeminiProvider` instance should update the `model_list` attribute correctly
- The `refresh_interval()` method of the `GeminiProvider` instance 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/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 LiteLLM providers incorrectly handle completion errors, causing incorrect annotations.

Key Assertions:

- The annotation should contain an error message indicating a completion error.
- The annotation should include the string 'boom' in its error message.
- The annotation should not contain any other error messages or strings that are not related to completion errors.
- The annotation should only contain one error message, even if there are multiple completion errors.
- The annotation should not raise a RuntimeError when it encounters a completion error, but instead return an error message.
- The annotation should include the correct type of error message (e.g. 'RuntimeError', 'ValueError', etc.)
- The annotation should be able to handle different types of completion errors (e.g. syntax errors, semantic errors, 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/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: This test prevents the provider from silently failing when receiving an invalid key_assertions payload.

Key Assertions:

- Ensure that the response data contains a list of key_assertions.
- Verify that the error message indicates a required field is missing.
- Check if the error message includes information about the invalid key_assertions.
- Test that the provider raises an AssertionError with a meaningful error message when receiving an invalid key_assertions payload.
- Ensure that the test case fails when the response data does not contain a list of key_assertions.
- Verify that the test case fails with a clear and descriptive error message.
- Check if the error message includes information about the specific validation rule being applied.
- Test that the provider raises an AssertionError with a meaningful error message when receiving an invalid key_assertions payload.

COVERAGE

src/pytest_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 annotates the missing dependency correctly.

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

Key Assertions:

- The annotation contains the correct error message indicating that 'litellm' is missing and how to install it.
- The annotation includes the correct dependency name ('litellm') in the error message.
- The annotation includes the correct URL for installing the dependency (pip install litellm).
- The annotation does not report an error when the dependency is installed correctly.
- The annotation does not report an error when the dependency is installed with a different package name (e.g., 'litello')
- The annotation does not report an error when the dependency has a different version number (e.g., '1.2.3' instead of '1.0.0')

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/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 correct key assertions and confidence level.

Why Needed: Prevents regression by ensuring the annotation is correctly configured for a successful response.

Key Assertions:

- status ok
- redirect
- confidence >= 0.8

COVERAGE

src/pytest_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 should return True when the 'litellm' module is available in the system's modules.
- The `is_available()` method should raise a `ModuleNotFoundError` if the 'litellm' module is not found in the system's modules.
- The `is_available()` method should correctly handle cases where multiple modules are installed with the same name (e.g., 'python', 'site-packages')
- The provider should be able to detect and report available modules that are not explicitly listed in the configuration (e.g., a package manager like pip)
- The provider should raise an error if the 'litellm' module is installed but not imported (i.e., it's not a direct import of the module)
- The provider should correctly handle cases where the system has multiple versions of the same module installed
- The provider should be able to detect and report available modules that are not installed using pip (e.g., a package manager like conda)

COVERAGE

src/pytest_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: Testing the `annotate_fallbacks_on_context_length_error` method in the `OllamaProvider` class.

Why Needed: This test prevents a potential regression where the method fails to annotate fallbacks when the context length error occurs.

Key Assertions:

- The `context_length_error` attribute of the provider is set to `True`.
- The `fallback_annotations` list contains at least one annotation.
- The `annotate_fallbacks_on_context_length_error` method returns a value that indicates success or failure.
- The `provider` object has an attribute `context_length_error` with value `True`.
- The `provider` object has an attribute `fallback_annotations` that is not empty.
- The `provider` object has an attribute `annotate_fallbacks_on_context_length_error` that returns a non-empty list of annotations.
- When the context length error occurs, the method annotates fallbacks 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	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: Test OllamaProvider::test_annotate_handles_call_error verifies that the annotate method handles call errors correctly.

Why Needed: This test prevents regression where the annotate method fails to handle call errors and returns an incorrect error message.

Key Assertions:

- The annotation should return 'Failed after 3 retries. Last error: boom' when a call error occurs.
- The annotation should not raise an exception if the call is successful.
- The annotation should provide a meaningful error message that indicates the cause of the failure.
- The annotation should handle multiple retries correctly (i.e., it should retry up to 3 times).
- The annotation should log any exceptions raised during the annotation process.
- The annotation should not fail if the system prompt is different from the user prompt.
- The annotation should provide a clear indication of whether the call was successful or failed.

COVERAGE

src/pytest_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: The Ollama provider should report an error when missing the required httpx dependency.

Why Needed: This test prevents a potential bug where the provider incorrectly reports a missing dependency without providing any useful information.

Key Assertions:

- assert annotation.error == 'httpx not installed. Install with: pip install httpx'
- assert mock_import_error('httpx') was called exactly once
- assert 'pip install httpx' in test_case.__doc__

COVERAGE

src/pytest_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 Ollama provider full annotation flow with mocked HTTP.

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

Key Assertions:

- check status
- validate token

COVERAGE

src/pytest_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: Ollama provider makes correct API call to generate text.

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

Key Assertions:

- The `url` captured is set to `http://localhost:11434/api/generate`.
- The `json` captured contains the expected model, prompt, system prompt, and stream values.
- The `timeout` captured is set to 60 seconds.
- The API call result matches the expected response string (`'test response'`).
- The `url` captured does not contain any additional parameters (e.g., `timeout`, `timeout_seconds`) that could prevent a successful API call.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Test that the Ollama provider uses the default model when not specified.

Why Needed: Prevents a regression where the default model is not used by the Ollama provider.

Key Assertions:

- The captured JSON response should contain the default model 'llama3.2'.
- The default model should be present in the captured JSON response even if it's empty.
- The default model should be used when no custom model is provided to the Ollama 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	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: Test that the Ollama provider returns False when the server is unavailable.

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

Key Assertions:

- The function _check_availability() of the OllamaProvider instance should return False.
- The function _check_availability() of the OllamaProvider instance should raise a ConnectionError when the server is unavailable.
- The provider should set its internal state to False after raising a ConnectionError.
- The provider's internal state should not be restored to its previous value before the test completes.
- The provider's internal state should only be set to False if the server is indeed unavailable.
- The provider's internal state should not be set to False if the server is already available.
- The provider's internal state should not be affected by other factors that may cause it to return a different 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/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.

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

Key Assertions:

- The method _check_availability() of the OllamaProvider instance returns False for non-200 status codes.
- The status code returned by the _check_availability() method is 500, which indicates a server error.
- The provider._check_availability() method should return False when the provided URL has an unknown or unhandled HTTP status 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/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: Test that the Ollama provider checks availability via /api/tags endpoint successfully.

Why Needed: To prevent a potential bug where the provider fails to check availability due to an incorrect or missing URL in the request.

Key Assertions:

- The provided URL '/api/tags' is present in the request.
- The response status code is 200 (OK).
- The Ollama provider correctly returns a list of available tags.
- The provider does not return any error messages or exceptions.
- The provider's availability check is performed successfully without any issues.

COVERAGE

src/pytest_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 incorrectly report its local status.

Key Assertions:

- provider.is_local() == True
- provider.provider_config['provider'] == 'ollama'
- is_local() is not 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/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 it encounters a non-JSON response from the Ollama API.

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

Key Assertions:

- The expected error message is 'Failed to parse LLM response as JSON'.
- The `annotation.error` attribute contains the correct error message.
- The `provider._parse_response()` method returns an instance of `Error` with the specified error message.
- The `provider._parse_response()` method raises a `ValueError` exception when it encounters a non-JSON response.
- The `Config` class is instantiated correctly with the provided provider name.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: The Ollama provider should raise an error when receiving a response with invalid 'key_assertions' payload.

Key Assertions:

- response_data
- oops
- Invalid response: key_assertions must be a list

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 extracts JSON from markdown code fences in a response.

Why Needed: This test prevents potential issues where the Ollama provider fails to extract JSON from code fences, potentially leading to incorrect or incomplete results.

Key Assertions:

- The extracted JSON is in the correct format and does not contain any unnecessary characters.
- The JSON contains only valid JSON syntax and does not include any invalid characters such as newlines or tabs.
- The JSON does not contain any nested objects or arrays that are not properly closed.
- Any Unicode characters within the JSON are properly escaped and do not cause any issues.
- The extracted JSON is a single object, rather than an array of objects.
- No errors are reported when parsing the JSON, indicating that it is valid.
- The JSON does not contain any duplicate keys or values that would cause conflicts with other data 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	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 parses a response in a plain Markdown fence without any language.

Why Needed: This test prevents regression where the provider fails to extract JSON from such responses, potentially leading to incorrect or incomplete model training data.

Key Assertions:

- The response is not empty.
- The response starts with an opening double backtick (``),
- The response ends with a closing double backtick (``).
- The response contains only whitespace characters and Markdown syntax, excluding any language tags or definitions.
- No JSON data is present within the response.
- The provider correctly identifies the presence of a plain fence without any language.

COVERAGE

src/pytest_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 successfully.

Why Needed: Prevents bugs in the Ollama provider by ensuring it correctly parses and extracts relevant information from 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: This test prevents a potential bug where the serialized representation of `CoverageEntry` is incorrect, potentially leading to unexpected behavior or errors in downstream applications.

Key Assertions:

- The 'file_path' key should contain the expected value.
- The 'line_ranges' key should contain the expected string representation.
- The 'line_count' key should contain the expected integer value.
- The 'coverage_data' key (if present) should be an empty dictionary.
- If `CoverageEntry` has a custom `__dict__` method, it should return the correct serialized representation.
- If `CoverageEntry` is missing any required keys, the test should fail with a clear 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: 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: Test that `CoverageEntry.to_dict()` correctly serializes a CoverageEntry object.

Why Needed: This test prevents the regression of coverage entry serialization issues.

Key Assertions:

- The 'file_path' key in the dictionary should match the expected value.
- The 'line_ranges' key in the dictionary should match the expected value.
- The 'line_count' key in the dictionary should match the expected value.
- The coverage entry's file path, line ranges, and line count are correctly serialized to a dictionary.
- Any additional keys present in the original CoverageEntry object are not included in the serialized dictionary.
- The order of the keys in the dictionary is preserved.
- The values of the key-value pairs in the dictionary match the expected values.
- No unexpected or extra keys are added to the dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: 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 set to the correct value.
- The 'line_ranges' key is set to the correct value.
- The 'line_count' key is set to the correct value.
- The coverage entry's file path matches the expected value.
- The coverage entry's line ranges match the expected values.
- The coverage entry's line count matches the expected value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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 potential bug where an empty annotation would have no confidence or error.

Key Assertions:

- The `annotation` object has an empty string for its `scenario` attribute.
- The `annotation` object has an empty string for its `why_needed` attribute.
- The `annotation` object is empty, and its `confidence` attribute is None.
- The `annotation` object is empty, and its `error` attribute is None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 `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 '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 data loss due to missing fields in the output.

Key Assertions:

- The 'scenario' field is present and matches the expected value.
- The 'confidence' field is set correctly within a specified range.
- The 'context_summary' field contains the required mode and bytes values.
- The 'error' field is absent, ensuring data integrity.
- All other fields are validated against their respective expectations.

COVERAGE

src/pytest_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

Why Needed: Prevents regression of default report schema version and empty test lists.

Key Assertions:

- The 'schema_version' key should be set to the current schema version.
- The 'tests' key should be an empty list.
- The 'warnings' key should not be included in the dictionary.
- The 'collection_errors' key should not be included in the dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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 reports with collection errors should be handled correctly.

Why Needed: This test prevents a potential bug where the report does not include all collection errors, potentially leading to false negatives.

Key Assertions:

- The length of `collection_errors` in the report dictionary should be exactly 1.
- The value at index 0 of `collection_errors` should have the correct nodeid.
- All nodes in `collection_errors` should have a non-empty 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 a ReportRoot instance is created with warnings.

Why Needed: To prevent a regression where the report does not include warnings, even when there are warnings.

Key Assertions:

- A ReportRoot instance is created with the specified warnings.
- The number of warnings in the report is exactly 1.
- The code of the first warning matches 'W001'.

COVERAGE

src/pytest_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: The test verifies that the `tests` list is sorted by `nodeid` in the output report.

Why Needed: This test prevents a regression where the sorting of tests by nodeid is not maintained across different test suites.

Key Assertions:

- The list of nodeids in the output should be in ascending order (a < m < z).
- The list of nodeids in the output should contain all nodes from the `tests` list.
- The list of nodeids in the output should not contain any duplicate nodeids.
- All nodes with a 'passed' outcome should have a corresponding test result in the sorted list.
- No nodeid should be missing from the sorted list.
- Node ids without an associated test result (i.e., `None` or empty string) should not be included in the sorted 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/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 a ReportWarning instance's 'detail' attribute is correctly populated with the provided path.

Why Needed: This test prevents a potential issue where the 'detail' attribute of a ReportWarning instance is missing, potentially leading to unexpected behavior or errors in downstream processing.

Key Assertions:

- The 'detail' key in the warning dictionary should contain the specified path.
- The value of the 'detail' key should be exactly '/path/to/file'.
- The 'detail' attribute of a ReportWarning instance is correctly populated with the provided 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/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: To prevent a warning about missing detailed information in the report.

Key Assertions:

- The 'detail' key is expected to be present in the dictionary.
- The value of the 'detail' key is not provided in this test.
- The 'code' and 'message' keys are already present in the dictionary with their correct values.
- Without detailed information, the report may contain misleading or incomplete 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	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: Verify that RunMeta has aggregation fields.

Why Needed: This test prevents regression where the aggregation policy is set to 'none' or 'simple'.

Key Assertions:

- The run_id field should be present in the meta dictionary.
- The run_group_id field should be present in the meta dictionary.
- The is_aggregated field should be True.
- The aggregation_policy field should be set to 'merge'.
- The run_count field should be equal to 3.
- The length of source_reports field should be equal to 2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	39 lines (ranges: 277-279, 281-283, 364-380, 382, 385, 387, 390, 393, 395, 397, 399-405, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 are disabled.

Why Needed: This test prevents regression where the LLM model's fields are still accessible even when annotations are disabled.

Key Assertions:

- llm_annotations_enabled should not be present in the data dictionary
- llm_provider should not be present in the data dictionary
- llm_model should not 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	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: Verify that LLM traceability fields are included when enabled for a RunMeta object.

Why Needed: This test prevents regression where the 'llm_annotations_enabled' field is missing or incorrectly set in a RunMeta object.

Key Assertions:

- The value of llm_annotations_enabled should be True.
- The value of llm_provider should match 'ollama'.
- The value of llm_model should match 'llama3.2:1b'.
- The value of llm_context_mode should match 'complete'.
- The value of llm_annotations_count should be 10.
- The value of 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 the `to_dict()` method of `RunMeta` to ensure it correctly excludes source reports when non-aggregated.

Why Needed: This test prevents a regression where source reports are included in aggregated report outputs.

Key Assertions:

- The 'source_reports' key is not present in the dictionary returned by `to_dict()`.
- The value of `is_aggregated` is set to `False` when `to_dict()` is called on a non-aggregated instance of `RunMeta`.
- When `to_dict()` is called on a non-aggregated `RunMeta` instance, it returns a dictionary with the correct keys and values.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	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 a potential bug where the 'git_sha' and 'plugin_git_sha' are not properly set in the test run meta.

Key Assertions:

- The 'git_sha' field should be set to 'abc1234'.
- The 'git_dirty' field should be set to True.
- The 'repo_version' field should be set to '1.0.0'.
- The 'repo_git_sha' field should be set to 'abc1234'.
- The 'repo_git_dirty' field should be set to True.
- The 'plugin_git_sha' field should be set to 'def5678'.
- The 'plugin_git_dirty' field should be set to False.
- The 'config_hash' field should be set to 'def5678'.

COVERAGE

src/pytest_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 that RunMeta includes run status fields in the JSON representation.

Why Needed: This test prevents a potential bug where the run status is not included in the output of `to_dict()` for certain types of RunMeta objects.

Key Assertions:

- The 'exit_code' field should be present and have value 1.
- The 'interrupted' field should be True.
- The 'collect_only' field should be True.
- The 'collected_count' field should be equal to the specified value (10).
- The 'selected_count' field should be equal to the specified value (8).
- The 'deselected_count' field should be equal to the specified value (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 that the schema version is correctly formatted as a semver string.

Why Needed: Prevents regression where the schema version is not in semver format, potentially causing issues with data validation or compatibility.

Key Assertions:

- The schema version should be split into three parts using '.' as the separator.
- Each part of the schema version should consist only of digits (0-9).
- All three parts of the schema version should have the same length (3 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/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 `ReportRoot` includes the correct schema version in its report root.

Why Needed: Prevents a potential bug where the schema version is not included in the report root, potentially causing issues with downstream processing or reporting.

Key Assertions:

- The `schema_version` attribute of `ReportRoot` should be equal to `SCHEMA_VERSION`.
- The value of `schema_version` as a string should also match `SCHEMA_VERSION`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	54 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestSourceCoverageEntry::test_to_dict 1ms 3

AI ASSESSMENT

Scenario: The test verifies that a CoverageEntry object can be serialized correctly into a dictionary.

Why Needed: This test prevents bugs or regressions where the coverage entry data is not properly converted to a dictionary.

Key Assertions:

- The 'file_path' key in the dictionary should match the expected value.
- The 'line_ranges' key in the dictionary should match the expected value.
- The 'line_count' key in the dictionary should match the expected value.
- The coverage entry data is correctly formatted and does not contain 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/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 regression where the minimal annotation is not serialized correctly without the 'confidence' field.

Key Assertions:

- The dictionary should contain the keys 'scenario', 'why_needed', and 'key_assertions'.
- The value of 'scenario' should be present in the dictionary.
- The value of 'why_needed' should be present in the dictionary.
- The value of 'key_assertions' should be present in the dictionary.
- The value of 'confidence' should not be present in the dictionary when it is None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	5 lines (ranges: 277-279, 281, 283)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestSourceReport::test_to_dict_with_run_id 1ms 3

AI ASSESSMENT

Scenario: Test SourceReport to_dict_with_run_id method with SourceReport object.

Why Needed: This test prevents a regression where the run_id is not included in the dictionary returned by to_dict().

Key Assertions:

- The 'run_id' key should be present in the dictionary.
- The value of the 'run_id' key should be 'run-1'.
- If SourceReport object does not have a run_id attribute, assert False.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: Test the `to_dict` method of CoverageEntry class.

Why Needed: This test prevents a potential bug where the coverage entry data is not properly serialized to JSON.

Key Assertions:

- The 'file_path' key in the dictionary should match the actual file path.
- The 'line_ranges' key in the dictionary should match the expected line ranges.
- The 'line_count' key in the dictionary should match the actual coverage count.
- Each assertion should be true, indicating that the data was successfully serialized to 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/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: Test that the `TestCaseResult` object has all required fields.

Why Needed: This test prevents a regression where the result is missing some critical information.

Key Assertions:

- The 'nodeid' field should be set to the expected value.
- The 'outcome' field should be set to 'passed'.
- The 'duration' field should be set to 0.0 (indicating no execution time).
- The 'phase' field should be set to 'call'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	17 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestTestCaseResult::test_result_with_coverage 1ms 3

AI ASSESSMENT

Scenario: tests/test_models.py::TestTestCaseResult::test_result_with_coverage verifies that the 'result' dictionary contains a single 'coverage' key.

Why Needed: This test prevents regression where coverage is not included in the result of tests with high code coverage.

Key Assertions:

- The 'coverage' list should contain exactly one entry.
- The 'file_path' of the first 'CoverageEntry' should match 'src/foo.py'.
- All 'CoverageEntry' values should have a 'file_path' attribute matching '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 the functionality of `TestCaseResult` with LLM opt-out.

Why Needed: Prevent regression in case LLM optimization is enabled.

Key Assertions:

- The 'llm_opt_out' key should be present and have a value of True.
- The 'llm_opt_out' key should only appear once per test result.
- If the 'llm_opt_out' key is missing, the test should fail.
- If the 'llm_opt_out' key has a different value than True, the test should fail.
- The 'llm_opt_out' key should be present in the dictionary representation of `TestCaseResult`.
- The 'llm_opt_out' key should only appear when it is set to True.
- If the 'llm_opt_out' key is not set at all, the test should fail.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 verifies that the `result` dictionary includes the `rerun_count` and `final_outcome` fields.

Why Needed: This test prevents a regression where the `result` dictionary is missing these critical fields when rerunning tests.

Key Assertions:

- The `rerun_count` field should be present in the `result` dictionary.
- The `final_outcome` field should also be present in the `result` dictionary.
- Both `rerun_count` and `final_outcome` fields should have the same value as expected (in this case, both are '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 'tests/test_models.py::TestTestCaseResult::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 a regression where the rerun count or final outcome is included in the result dictionary when running tests without reruns.

Key Assertions:

- The 'rerun_count' key should be absent from the `result` dictionary.
- The 'final_outcome' key should be absent from the `result` dictionary.
- The `result` dictionary should not contain any keys other than 'nodeid', 'outcome', and 'final_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/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 test_default_values scenario.

Why Needed: This test prevents a regression where default values might not be set correctly, potentially leading to unexpected behavior or errors in the application.

Key Assertions:

- cfg.provider == 'none'
- cfg.llm_context_mode == 'minimal'
- cfg.llm_max_tests == 0
- cfg.llm_max_retries == 3
- cfg.llm_context_bytes == 32000
- cfg.llm_context_file_limit == 10
- cfg.llm_requests_per_minute == 5
- cfg.llm_timeout_seconds == 30
- cfg.llm_cache_ttl_seconds == 86400
- cfg.include_phase == 'run'
- cfg.aggregate_policy == 'latest'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_options.py::TestConfig::test_get_default_config 1ms 3

AI ASSESSMENT

Scenario: Verify that the default configuration is correctly returned by the `get_default_config` method.

Why Needed: This test prevents a potential bug where the default configuration is not correctly set to 'none'.

Key Assertions:

- The function `get_default_config()` returns an instance of `Config`.
- The attribute `provider` on the returned `Config` object is set to `''none''`.
- A valid default configuration should be returned for the `provider` 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	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 when switching between OLLAMA and NONE provider.

Key Assertions:

- The function should return False when provider is 'none'.
- The function should return True when provider is 'ollama'.
- The function should not return True when provider is set to 'ollama' before calling `is_llm_enabled()`.
- The function should throw an error when provider is set to 'ollama' without setting the `provider` attribute first.

COVERAGE

src/pytest_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 validation with an invalid provider.

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

Key Assertions:

- The configuration object should have exactly one error message.
- The error message should contain 'Invalid provider 'invalid_provider'.
- The error message should be present in the list of errors returned by validate().

COVERAGE

src/pytest_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 TestConfig.

Why Needed: Prevents regression where the llm_context_bytes is set to a value less than 1000, which may cause invalid LLM context bytes.

Key Assertions:

- cfg.validate() returns an error with message 'llm_context_bytes must be at least 1000'
- The 'llm_context_bytes' field in the configuration does not have any default value
- The validation of numeric constraints for llm_context_bytes is performed correctly
- No errors are returned when llm_context_bytes is set to a valid value (e.g., 500)
- The error message 'llm_context_bytes must be at least 1000' is present in the list of errors

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 issues.

Why Needed: This test prevents potential bugs where an invalid configuration is passed to the validate method.

Key Assertions:

- The validate method of the Config object does not return any error when a valid config is provided.
- A ValueError or other exception is not raised when a valid config is passed to the validate method.
- The validate method checks for missing required parameters and raises an error if they are missing.
- The validate method checks for invalid values of required parameters and raises an error if they are invalid.
- The validate method checks for duplicate keys in the configuration and raises an error if a key is duplicated.
- The validate method checks for circular dependencies between configuration options and raises an error if a circular dependency is detected.
- The validate method checks for missing or extra required parameters in the configuration and raises an error if they are missing or extra.
- The validate method checks for invalid values of optional parameters and raises an error if they are 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	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 correct aggregate directory, policy and run ID.

Why Needed: This test prevents a regression where the aggregation options are not correctly loaded due to an incorrect or missing aggregate directory.

Key Assertions:

- The `aggregate_dir` option is set to `aggr_dir`.
- The `aggregate_policy` option is set to `merge`.
- The `aggregate_run_id` option is set to `run-123`.
- The `aggregate_group_id` option is set to `group-abc`.
- The aggregate directory, policy and run ID are correctly loaded from the configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 the handling of invalid integer values in INI configuration files.

Why Needed: This test prevents a potential regression where the program crashes when encountering an invalid integer value in the INI file.

Key Assertions:

- The function load_config() should not crash or throw an exception when it encounters an invalid integer value in the INI file.
- The default value of llm_max_retries is set to 3, which is a reasonable fallback value for most cases.
- The mock_pytest_config.getini.side_effect is set to a lambda function that returns the ini_values dictionary, allowing the test to verify the expected 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	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: Verify that the `llm_coverage_source` option is set to 'cov_dir' after loading the configuration.

Why Needed: This test prevents a potential bug where the coverage source is not correctly set even if the `llm_coverage_source` option is provided.

Key Assertions:

- The value of `cfg.llm_coverage_source` should be equal to 'cov_dir' after calling `load_config(mock_pytest_config)`.
- The configuration object `cfg` has an attribute `llm_coverage_source` with the expected value 'cov_dir'.
- The `mock_pytest_config.option.llm_coverage_source` attribute is set to 'cov_dir' before calling `load_config(mock_pytest_config)`.
- The `load_config(mock_pytest_config)` function correctly loads the configuration and sets the `llm_coverage_source` option.
- The `cfg.llm_coverage_source` attribute is updated with the correct value after loading the configuration.
- The test case passes if the `llm_coverage_source` option is set to 'cov_dir' in the loaded 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	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: Verify that the `load_defaults` test loads a default provider and report HTML to an empty configuration.

Why Needed: Prevents regression when no options are set for pytest.

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 a potential regression where the default value of llm_report_html is overridden by cli_report.html.

Key Assertions:

- cfg.report_html should be set to 'cli_report.html' as expected.
- cfg.llm_requests_per_minute should not be set to 100 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	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 set to 9 when loading from CLI.

Why Needed: This test prevents a bug where the `llm_max_retries` option is not correctly updated when loading configuration from the command line.

Key Assertions:

- The value of `llm_max_retries` in the loaded configuration should be 9.
- The `llm_max_retries` option should be set to a non-negative integer (in this case, 9).
- The `load_config` function should correctly update the `llm_max_retries` option when loading from CLI.
- The `mock_pytest_config.option.llm_max_retries` attribute should have been set to 9 before calling `load_config`.

COVERAGE

src/pytest_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 Pytest configuration.

Why Needed: Prevents a potential bug where the test fails if the `llm_report_provider` option is not set in the ini file.

Key Assertions:

- The `provider` attribute of the configuration object should be 'ollama'.
- The `model` attribute of the configuration object should be 'llama3'.
- The `context_mode` attribute of the configuration object should be 'balanced'.
- The `requests_per_minute` attribute of the configuration object should be 10.
- The `max_retries` attribute of the configuration object should be 5.
- The `html` attribute of the configuration object should be set to 'report.html'.
- The `json` attribute of the configuration object 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 configuration of aggregation settings to ensure they match expected values.

Why Needed: This test prevents a potential bug where the aggregation policy or include history are not correctly configured.

Key Assertions:

- The `aggregate_dir` attribute is set to `/reports` as expected.
- The `aggregate_policy` attribute is set to `merge` as expected.
- The `aggregate_include_history` attribute is set to `True` 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	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 a potential bug where the report configuration is not correctly set for multiple output formats.

Key Assertions:

- config.report_html == 'report.html'
- config.report_json == 'report.json'
- config.report_pdf == 'report.pdf'
- config.report_evidence_bundle == 'bundle.zip'
- config.report_dependency_snapshot == 'deps.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`	`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 `capture_failed_output` is set to `True` in the test configuration.

Why Needed: This test prevents a potential issue where the test fails due to an incorrect or missing capture settings.

Key Assertions:

- The `capture_failed_output` attribute of the test configuration should be `True`.
- The `capture_output_max_chars` attribute of the test configuration should be set to 8000.
- The test will fail if `capture_failed_output` is not `True` or `capture_output_max_chars` is not 8000.
- The capture settings are correctly configured for the test.
- The test configuration does not contain any missing or incorrect capture settings.

COVERAGE

src/pytest_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 correctly set in the configuration.

Why Needed: This test prevents a bug where the compliance settings are not properly configured, potentially leading to incorrect or missing metadata.

Key Assertions:

- The value of `config.metadata_file` is 'metadata.json'.
- The value of `config.hmac_key_file` is '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: This test prevents a bug where coverage settings are not correctly configured, potentially leading to incorrect reporting or missed tests.

Key Assertions:

- config.omit_tests_from_coverage is False
- config.include_phase == 'all'
- asserts that the configuration includes all phases (in this case, all)
- asserts that omitting tests from coverage is not enabled 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/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: Test the configuration of LLM context excluding certain file extensions.

Why Needed: This test prevents a potential bug where custom exclude globs are not properly propagated to the LLM context.

Key Assertions:

- The function `Config(llm_context_exclude_globs)` is called with the specified list of excluded glob patterns.
- The assertion `*.pyc` is present in the list of excluded glob patterns.
- The assertion `*.log` is not present in the list of excluded glob patterns (or it should be, depending on the expected 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	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: Test the ability to specify include globs for LLM context.

Why Needed: This test prevents a bug where the include globs are not correctly applied to the LLM context.

Key Assertions:

- The string `*.py` is in the list of include globs.
- The string `*.pyi` is in the list of include globs.
- The string `*.class` is included in the list of include globs (not tested).
- The correct order of include globs is maintained (e.g., `*.py` before `*.pyi`).

COVERAGE

src/pytest_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

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	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: Tests the configuration of LLM execution settings.

Why Needed: Prevents regression in LLM execution settings configuration.

Key Assertions:

- The value of llm_max_tests should be 50.
- The value of llm_max_concurrency should be 8.
- The value of llm_requests_per_minute should be 12.
- The value of llm_timeout_seconds should be 60 seconds.
- The value of llm_cache_ttl_seconds should be 3600 seconds.
- The directory for the LLM cache should be .cache.

COVERAGE

src/pytest_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 number of characters for LLM parameter values is not checked.

Key Assertions:

- config.llm_include_param_values should be True
- config.llm_param_value_max_chars should equal 200
- assert config.llm_include_param_values is True checks if it's indeed True
- assert config.llm_param_value_max_chars == 200 checks if it's not less than or equal to 200

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_llm_settings` 1ms 3

AI ASSESSMENT

Scenario: Tests the configuration of LLM settings for the ollama provider.

Why Needed: This test prevents a potential bug where the model name is not correctly set to 'llama3.2'.

Key Assertions:

- assert config.provider == "ollama",
- assert config.model == "llama3.2",
- assert config.llm_context_bytes == 64000

COVERAGE

src/pytest_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 of the `Config` object is set to `/project`.

Why Needed: This test prevents a potential bug where the repository root path is not correctly set when creating a new `Config` instance with a custom `repo_root` parameter.

Key Assertions:

- config.repo_root
- is equal to Path('/project')
- config.repo_root is set before any other operations in this test method

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: Test the test_valid_phase_values function to ensure all valid include_phase values pass validation.

Why Needed: This test prevents regression by verifying that only valid include_phase values are included in the configuration.

Key Assertions:

- The 'include_phase' key should not be present in any error messages for valid include_phase values.
- Only valid include_phase values (e.g. 'run', 'setup', 'teardown', 'all') should be reported as errors.
- Any invalid or missing include_phase value should result in no error message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	17 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_exclude_globs` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the default exclude globs are correctly set to include `*.pyc`, `__pycache__/*`, and specific paths containing certain keywords.

Why Needed: This test prevents a potential regression where the default exclude globs do not include important files like `.pyc` or `__pycache__/*`.

Key Assertions:

- The function `defaults = config.llm_context_exclude_globs` returns a list of strings that includes `*.pyc`, `__pycache__/*`, and specific paths containing certain keywords.
- The string `*.pyc` is present in the returned list of exclude globs.
- The string `__pycache__/*` is present in the returned list of exclude globs.
- The string `*secret*` is present in the returned list of exclude globs.
- The string `*password*` is present in the returned list of exclude globs.

COVERAGE

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

Why Needed: Prevents a potential bug where the default redact patterns are not correctly identified or included in the test configuration.

Key Assertions:

- The `--password` and `--token` flags should be present in the default redact patterns.
- The `--api[_-]?key` flag should be present 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 TestConfigDefaultsMaximal class.

Why Needed: This test prevents a potential regression where the default provider and llm context mode are not set correctly.

Key Assertions:

- The `provider` attribute of the config object should be set to 'none'.
- The `llm_context_mode` attribute of the config object should be set to 'minimal'.
- The `llm_context_bytes` attribute of the config object should be set to 32000.
- The `omit_tests_from_coverage` attribute of the config object should be set to True.
- The `include_phase` attribute of the config object 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 correct enabled status for different providers.

Why Needed: Prevents a potential regression where the `is_llm_enabled` method might return incorrect results due to changes in provider configuration.

Key Assertions:

- The function should return False when the provider is 'none'.
- The function should return True when the provider is 'ollama' or 'litellm'.

COVERAGE

`src/pytest_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: Prevent a potential bug where an invalid aggregate policy is passed to the Config class, potentially causing unexpected behavior or errors.

Key Assertions:

- The Config instance should have one error associated with it.
- The error message should contain 'Invalid aggregate_policy 'invalid''.
- The error message should be present in the first error found by the validate() method.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-197, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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

Scenario: Test validates an invalid include phase.

Why Needed: Prevents a potential bug where the test incorrectly handles an invalid include phase.

Key Assertions:

- The config object is created with an invalid include phase.
- An error message indicating the invalid include phase is returned.
- The error message includes the specific phrase 'invalid'.
- A single error is reported for the invalid include phase.
- The test asserts that only one error is found in the configuration.
- The error message does not contain any additional information about the include phase.
- The test verifies that the error is related to the include phase.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-205, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 unexpected error message for an invalid provider.

Key Assertions:

- The function `validate()` should return exactly one error.
- The error message should contain 'Invalid provider 'invalid''.
- The error message should be present in the first error found.
- The test should fail when an invalid provider is provided.
- The error should not be a generic 'Invalid input' message but rather specific to the invalid provider.
- The error message should include the actual provider name ('invalid')
- The error message should not contain any additional information that could cause confusion (e.g., 'Provider XYZ')

COVERAGE

`src/pytest_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: This test prevents a bug where the config is not validated correctly when it contains invalid numeric values.

Key Assertions:

- The 'llm_context_bytes' field should be an integer.
- The 'llm_max_tests' field should be an integer.
- The 'llm_requests_per_minute' field should be an integer.
- The 'llm_timeout_seconds' field should be an integer.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 a valid configuration returns an empty list.

Why Needed: Prevents invalid configurations from being created and causing unexpected behavior.

Key Assertions:

- The `validate()` method of the Config object should return an empty list when a valid config is provided.
- A valid config should not raise any exceptions or errors during validation.
- The `validate()` method should only return an empty list for valid 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: Test that the configuration defaults to safe settings when no options are provided.

Why Needed: Prevents a potential bug where the plugin's default configuration is not set correctly without explicit options being passed.

Key Assertions:

- The `cfg` variable should be an instance of `Config`.
- The `cfg` variable should have safe defaults.
- The `cfg` variable can't be checked for exact values since pytest may not have our options registered.

COVERAGE

src/pytest_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 the `pytestconfig` object is accessible in the test.

Why Needed: This test prevents a potential bug where the plugin configuration is not properly loaded.

Key Assertions:

- The `pytestconfig` object should be an instance of `pytest.config.Config` or `None`.
- The `pytestconfig` object should have attributes such as `name`, `version`, and `format`.
- The `pytestconfig` object should be accessible through the `pytest` module.
- The `pytestconfig` object should not be `None` when accessed.
- The `pytestconfig` object's attributes should be correctly set.
- The `pytestconfig` object's format attribute should be a string.
- The `pytestconfig` object's name attribute should be a 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)

PASSED `tests/test_plugin_integration.py::TestPluginIntegration::test_llm_context_marker` 1ms 2

AI ASSESSMENT

Scenario: The test verifies that the context marker does not cause errors.

Why Needed: This test prevents a bug where the LLM context marker causes an error in the plugin integration.

Key Assertions:

- The `test_llm_context_marker` function should pass without any errors.
- The output of the `pytest_llm_report/collector.py` module should not contain any error messages.
- The `src/pytest_llm_report/plugin.py` module should not raise any exceptions during execution.

COVERAGE

src/pytest_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 test verifies that the requirement marker does not cause any errors.

Why Needed: This test prevents a potential bug where the requirement marker might be misinterpreted or cause an error in some scenarios.

Key Assertions:

- The function `test_requirement_marker` is called without any arguments.
- The variable `requirement_marker` is assigned a value using the `assert` keyword.
- No errors are raised when calling `test_requirement_marker`.
- The test passes if no exception is thrown during execution of `test_requirement_marker`.
- The function does not throw any exceptions or raise an error.
- The variable `requirement_marker` remains unchanged after the call to `test_requirement_marker`.
- No changes are made to the code that might affect its functionality.
- The test does not rely on external dependencies for execution.

COVERAGE

src/pytest_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` 33ms 6

AI ASSESSMENT

Scenario: Test the integration of report writer with pytest_llm_report.

Why Needed: This test prevents regression that may occur when using a custom report writer.

Key Assertions:

- Verify that the report writer writes a JSON file containing the correct summary statistics.
- Verify that the HTML file contains references to all test files.
- Check if the report writer correctly identifies test cases as passed or failed with an error message.
- Ensure that the total count of tests is accurate in both JSON and HTML reports.
- Verify that the report writer writes a valid HTML file with links to test files.
- Test that the report writer can handle multiple test cases with different outcomes (passed, failed, and 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/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 collectreport is disabled.

Why Needed: This test prevents a regression where collectreport is disabled and pytest_collectreport still runs.

Key Assertions:

- The `pytest_collectreport` function should not be called with the `_enabled_key` key when `collectreport` is disabled.
- The `get` method of `session.config.stash` should return False for the specified key.
- The `assert_called_with` method of `mock_report.session.config.stash.get` should not be called.
- The `pytest_collectreport` function should not have been called with the `_enabled_key` key when `collectreport` is disabled.
- The `get` method of `session.config.stash` should return False for the specified key.
- The `assert_called_with` method of `mock_report.session.config.stash.get` should not be called.
- The `pytest_collectreport` function should not have been called with the `_enabled_key` key when `collectreport` is disabled.
- The `get` method of `session.config.stash` should return False for the specified key.
- The `assert_called_with` method of `mock_report.session.config.stash.get` should not be called.

COVERAGE

src/pytest_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 it should, and to ensure the collector is properly registered with pytest_collectreport.

Key Assertions:

- The stash_get function returns True for _enabled_key and mock_collector.
- The stash_get function returns mock_collector for _collector_key.
- mock_collector.handle_collection_report is called once with mock_report as argument.
- mock_collector.handle_collection_report does not call stash_get again after the first call.
- The stash_get function does not return any value when _enabled_key or _collector_key are not found.
- The stash_get function returns a default value (None) for key that is not found in config.
- The stash_get function does not raise an exception if it cannot find a 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`	`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 throw an exception when a `session` is not available.

Why Needed: Prevents the plugin from skipping the collect report due to a missing session.

Key Assertions:

- The function `pytest_collectreport(mock_report)` should not be called with a `session` attribute that is `None` or an empty object.
- The function `pytest_collectreport(mock_report)` should raise a `AttributeError` when trying to access the `session` attribute.
- The `session` attribute of the mock report object should be `None` or an empty object before calling `pytest_collectreport()`.
- After calling `pytest_collectreport(mock_report)`, the `session` attribute of the mock report object should still be `None` or an empty object.
- If a session is available, `pytest_collectreport(mock_report)` should not raise an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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 `pytest_collectreport` does not throw an exception when the session is set to `None`.

Why Needed: This test prevents a potential bug where `pytest_collectreport` would raise an error when trying to access the `session` attribute of a mock report object with `None` as its value.

Key Assertions:

- The `pytest_collectreport` function should not throw any exceptions when given a `mock_report` object with `None` as its `session` attribute.
- The `session` attribute of the `mock_report` object is set to `None` before calling `pytest_collectreport`.
- After calling `pytest_collectreport`, the `session` attribute of the `mock_report` object remains unchanged and does not become `None`.
- A test case with a `None` session should not cause any errors or unexpected behavior in the application.
- The `pytest_collectreport` function is designed to handle such scenarios without raising an exception.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/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 configuring pytest_llm_report plugin.

Why Needed: This test prevents a potential bug where the LLM report provider 'ollama' is enabled without being configured properly, leading to an unhandled warning.

Key Assertions:

- The `pytest_llm_report_provider` option should be set to `'ollama'` for LLM report.
- The `llm_report_html`, `llm_report_json`, and `llm_report_pdf` options should not be set to `None` when the `llm_report_provider` is set to `'ollama'`.
- The `llm_evidence_bundle`, `llm_dependency_snapshot`, `llm_requests_per_minute`, `llm_aggregate_dir`, `llm_aggregate_policy`, and `llm_aggregate_run_id` options should not be set to `None` when the `llm_report_provider` is set to `'ollama'`.
- The `llm_max_retries` option should have a valid value.
- The `rootpath` option should point to a valid directory.
- The `stash` option should be an empty dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 invalid configuration is provided.

Why Needed: To prevent a UsageError from being raised due to an invalid configuration in the pytest_llm_report.plugin.pytest_configure function.

Key Assertions:

- mock_config.option.llm_report_html is None
- mock_config.option.llm_report_json is None
- mock_config.option.llm_report_pdf is None
- mock_config.option.llm_evidence_bundle is None
- mock_config.option.llm_dependency_snapshot is None
- mock_config.option.llm_requests_per_minute is None
- mock_config.option.llm_aggregate_dir is None
- mock_config.option.llm_aggregate_policy is None
- mock_config.option.llm_aggregate_run_id is None
- mock_config.option.llm_aggregate_group_id is None
- mock_config.rootpath is not a 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/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.

Why Needed: This test prevents a potential regression where the plugin might not skip configuration due to an issue with worker input.

Key Assertions:

- mock_config.addinivalue_line was called before pytest_configure was called.
- addinivalue_line is still called for markers before worker check
- pytest_configure was not called because addinivalue_line was already called

COVERAGE

src/pytest_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.

Why Needed: Prevents a potential bug where the plugin does not load configuration due to missing Config.load method.

Key Assertions:

- mock_load.assert_called_once()
- mock_cfg.validate.return_value == []
- pytest_configure(mock_config) should be called with mock_config
- mock_load.return_value is None
- mock_cfg.getini.return_value is None
- mock_cfg.option.llm_report_html is None
- mock_cfg.option.llm_max_retries 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	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 for the plugin.

Why Needed: This test prevents a potential regression where setting CLI options (llm_report_html, llm_report_json, etc.) to None would prevent the plugin from loading its configuration correctly.

Key Assertions:

- The provider of the plugin is set to 'ollama'.
- The model used by the plugin is set to 'llama3.2'.
- The context mode for the plugin is set to 'complete'.
- The number of requests per minute for the plugin is set to 10.
- The HTML report file name is set to 'ini.html'.
- The JSON report file name is 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 a potential bug where the CLI options override INI options, causing unexpected behavior when running the plugin with configuration from both sources.

Key Assertions:

- The 'llm_report_html' option should be set to 'cli.html'.
- The 'llm_report_json' option should be set to 'cli.json'.
- The 'llm_report_pdf' option should be set to 'cli.pdf'.
- The 'llm_evidence_bundle' option should be set to 'bundle.zip'.
- The 'llm_dependency_snapshot' option should be set to 'deps.json'.
- The 'llm_requests_per_minute' option should be set to 20.
- The 'aggregate_dir' option should be set to '/agg'.
- The 'aggregate_policy' option should be set to 'merge'.
- The 'aggregate_run_id' option should be set to 'run-123'.
- The 'aggregate_group_id' option should be set to 'group-abc'.
- The rootpath is correctly set to '/project'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: To prevent a regression where the plugin's terminal summary is not properly handled when it is disabled.

Key Assertions:

- mock_config.stash.get.assert_called_once_with(_enabled_key, False)
- assert result is None
- assert mock_config.stash.get.return_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/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 no configuration is provided.

Why Needed: Prevents regression in case the test is run without specifying an xdist worker.

Key Assertions:

- The function pytest_terminal_summary should return None for a mock config object with no 'workerid' key.
- The function pytest_terminal_summary should not perform any actions on the mock config object.
- The function pytest_terminal_summary should assert that it returns None without doing anything.
- The function pytest_terminal_summary should not raise an exception or throw an error when called with a mock config object.
- The function pytest_terminal_summary should return early without doing anything in this case.
- The function pytest_terminal_summary should be able to handle the mock config object correctly even if it has no 'workerid' 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	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 the correct value is used for `report_html` option.

Why Needed: This test prevents a potential bug where the `report_html` option is not set correctly, leading to incorrect output in the report.

Key Assertions:

- mock_config.option.llm_report_html == 'out.html'
- cfg.report_html == 'out.html'
- mock_config.rootpath == '/root'
- cfg.rootpath == '/root'
- cfg.getini('llm_report_html') == None
- cfg.getini('llm_report_json') == 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	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's report generation is not executed when makereport is disabled.

Key Assertions:

- mock_item.config.stash.get() returns False
- mock_call.call() does not yield any value (i.e., no result)
- mock_outcome.get_result().get_result() returns None
- gen.send(mock_outcome) raises StopIteration

COVERAGE

`src/pytest_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 that makereport calls collector when enabled.

Why Needed: Prevents a potential bug where the plugin does not collect and report test results even if makereport is enabled.

Key Assertions:

- The `pytest_runtest_makereport` function should be able to find and call the `mock_collector` when it is set as the collector.
- The `mock_collector.handle_runtest_logreport` method should be called with the correct arguments (the report and the item).
- The `mock_collector` should not be used in a way that prevents makereport from being enabled or configured correctly.
- The `pytest_runtest_makereport` function should be able to yield control back to the test runner when it is no longer needed (in this case, by calling `next(gen)`).
- The `mock_collector.handle_runtest_logreport` method should not raise an exception when called with a mock report.
- The `pytest_runtest_makereport` function should be able to handle the case where the collector is set but makereport is disabled (i.e., the `mock_collector` is not used).
- The test should pass without any errors or exceptions being raised.

COVERAGE

`src/pytest_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 and pytest_collection_finish function is called.

Why Needed: This test prevents a regression where the plugin's hooks are not properly executed 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 mock_session
- mock_session.config.stash.get.return_value should be set to False
- pytest_collection_finish(mock_session) should not have any side effects

COVERAGE

src/pytest_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 `pytest_collection_finish` calls the collector when collection_finish is enabled.

Why Needed: This test prevents a potential regression where `pytest_collection_finish` does not call the collector even if it's enabled.

Key Assertions:

- The `stash_get` function returns the correct value for `_enabled_key` and `mock_collector` keys.
- The `handle_collection_finish` method of `mock_collector` is called with the correct arguments.
- The `items` attribute of `mock_session` is modified correctly before calling `pytest_collection_finish`.

COVERAGE

`src/pytest_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 verifies the expected behavior.

Why Needed: To prevent a potential bug where pytest_sessionstart fails to check the enabled status of the plugin.

Key Assertions:

- mocked config stash.get for _enabled_key should have been called with False as argument.
- pytest_sessionstart mock session should have been called with False as argument.
- mocked config stash.get for _enabled_key should not have been called again.
- pytest_sessionstart mock session should not be called again.
- mocked config stash.get for _enabled_key should not have been called with True as argument.
- pytest_sessionstart mock session should not have been called with False as 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`	`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.

Why Needed: Prevents a potential bug where the collector is not initialized due to an unenabled sessionstart setting.

Key Assertions:

- The _collector_key should be present in the mock_stash dictionary.
- The _start_time_key should also be present in the mock_stash dictionary.
- The collector should have been created successfully.
- _collector_key and _start_time_key should both exist in the stash_dict.
- The stash_dict should contain a True value for _enabled_key.
- The stash_dict should contain a Config object for _config_key.
- The stash_dict should contain a dictionary with a key-value pair for _enabled_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	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 to the parser.

Why Needed: pytest_addoption may not correctly handle all cases where LLM-enhanced test reports are requested with specific options like --llm-report or --llm-coverage-source.

Key Assertions:

- parser.getgroup.assert_called_with('llm-report', 'LLM-enhanced test reports')
- group.addoption.call_args_list[0][0] == '--llm-report'
- group.addoption.call_args_list[1][0] == '--llm-coverage-source'

COVERAGE

src/pytest_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: Verifies that pytest_addoption adds INI options (lines 13-34) to the plugin.

Why Needed: This test prevents a regression where pytest_addoption does not add INI options.

Key Assertions:

- Verify that 'llm_report_html' is added as an ini option.
- Verify that 'llm_report_json' is added as an ini option.
- Verify that 'llm_report_max_retries' is added as an ini 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/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 when using the terminal summary feature.

Key Assertions:

- The `report_html` option is set to 'out.html'.
- A mock Coverage object is created and its `load` method is called with a configuration stash.
- The `report` method of the mock Coverage object is called once.
- The `CoverageMapper` class is patched to return a mock Coverage object.
- The `ReportWriter` class is patched to call the `report` method on the mock Coverage object.
- The `MockStash` dictionary is created and its `stash` attribute is set to the mock stash.
- The `MagicMock` instance is created with the mock stash as its stash attribute.
- The `pytest_terminal_summary` function is called with a mock instance, 0 iterations, and the mock config stash.

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 plugin functionality when LLM is enabled.

Key Assertions:

- Verify that the `pytest_terminal_summary_llm_enabled` test passes without any errors.
- Confirm that the `pytest_terminal_summary_llm_enabled` test correctly sets up a mock stash with the provided configuration.
- Check that the `pytest_terminal_summary_llm_enabled` test patches dependencies at source to return a mock writer.
- Verify that the `pytest_terminal_summary_llm_enabled` test annotates tests with the correct configuration.
- Ensure that the `pytest_terminal_summary_llm_enabled` test passes without any errors or regressions in the plugin.
- Confirm that the `pytest_terminal_summary_llm_enabled` test correctly sets up a mock provider with the specified model name.
- Verify that the `pytest_terminal_summary_llm_enabled` test returns the expected configuration for the annotation.

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 if missing.

Why Needed: This test prevents a regression where the plugin does not create a collector when it is not enabled.

Key Assertions:

- The stash should be empty before calling pytest_terminal_summary().
- The stash should contain only _enabled_key and _config_key after calling pytest_terminal_summary().
- The stash should have both get() and [] methods available for the stash object.
- The stash should not have any other keys when it is created.
- The stash should be empty before calling pytest_terminal_summary().
- The stash should contain only _enabled_key and _config_key after calling pytest_terminal_summary().
- The stash should have a valid type of dict.
- The stash should have both get() and [] methods available for the stash object.

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 the case where aggregation is disabled and terminal summary is not aggregated correctly.

Key Assertions:

- The aggregate_dir parameter should be set to '/agg' when aggregation is enabled.
- The stash object should support both get() and [] methods.
- The aggregation method of the Aggregator class should return a report.
- The ReportWriter class should write JSON and HTML files correctly when called with the correct arguments.
- The aggregate function of the Aggregator class should be called once during the test execution.
- The stash object should not have any worker input properties.
- The stash object should support aggregation for both get() and [] methods.

COVERAGE

src/pytest_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

Scenario: Test coverage calculation error when loading coverage map during terminal summary.

Why Needed: This test prevents a regression where the coverage calculation fails due to an OSError being raised during load of the coverage map.

Key Assertions:

- The `coverage.load` method should not raise an exception if it is successful.
- The `coverage.CoverageMapper` instance created from the loaded coverage map should be valid and functional.
- The `pytest_llm_report.coverage_map.CoverageMapper` class should be able to load the coverage map without raising a `FileNotFoundError`.
- The `pytest_terminal_summary` function should not raise a `UserWarning` when called with an invalid configuration.
- The `MagicMock` instance created for testing purposes should have the correct attributes and methods.
- The `report_writer` attribute of the mock `ReportWriter` instance should be set to a valid object.
- The `coverage.Coverage` instance created from the loaded coverage map should have the correct attributes and methods.
- The `MagicMock` instance created for testing purposes should not raise an exception when called with a valid configuration.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	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 with a test file and dependency.

Why Needed: This test prevents regression when assembling contexts with unbalanced dependencies, as it ensures that all required files are included.

Key Assertions:

- The 'utils.py' file is present in the assembled context.
- The 'def util()' function is found in the 'utils.py' file.
- All lines from 'utils.py' are included in the assembled context.
- No other files are included in the assembled context except for 'utils.py'.
- The 'test_1()' function is present in the assembled context.
- The 'def test_1()' function is found in the 'utils.py' 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/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: Test the `ContextAssembler` class to assemble a complete context for a test file.

Why Needed: This test prevents regression when the `ContextAssembler` is used with a configuration that requires a 'complete' llm_context_mode.

Key Assertions:

- The assembly result contains the name of the function being tested (in this case, `test_1`).
- The source code of the test file is included in the assembled context.
- The function being tested has an outcome of 'passed'.
- The `ContextAssembler` correctly assembles a complete context for the test 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/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 a bug where the ContextAssembler fails to assemble a minimal context for a test file with a single test function, resulting in an incorrect test result.

Key Assertions:

- The source code of the test function is present in the assembled context.
- The assembled context has no additional information besides the test function.
- The test function is correctly identified as part of the assembled context.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Test the ContextAssembler with balanced context limits to ensure it correctly truncates long files and reports coverage.

Why Needed: This test prevents regression in the ContextAssembler's behavior when dealing with large files that exceed the default context limit.

Key Assertions:

- The 'f1.py' file is present in the assembled context.
- ... truncated message is included in the assembled context.
- Length of the assembled context does not exceed 40 bytes (20 bytes + truncation message).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Verifies that the ContextAssembler correctly handles non-existent files and nested test names with parameters.

Why Needed: This test prevents a potential bug where the assembler incorrectly returns an empty string for a non-existent file.

Key Assertions:

- The function `_get_test_source` is called with a valid path `none.py::test` and it should return an empty string.
- The function `_get_test_source` is called with a valid path `test_p.py::test_param[1]` and it should return the correct 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	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 bug where the ContextAssembler incorrectly excludes certain files, potentially leading to incorrect results or errors.

Key Assertions:

- config._should_exclude('*.pyc') is True
- config._should_exclude('secret/*') 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 numbers without gaps are not compressed into ranges.

Key Assertions:

- assert compress_ranges([1, 2, 3]) == '1-3'
- assert compress_ranges([4, 5, 6]) == '4-6'
- assert compress_ranges([7, 8, 9]) == '7-9'
- assert compress_ranges([]) == ''
- assert compress_ranges([1]) == '1'

COVERAGE

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

Why Needed: This test prevents a potential bug where the function incorrectly identifies non-duplicate ranges as duplicates.

Key Assertions:

- assert compress_ranges([1, 2, 2, 3, 3, 3]) == '1-3'
- assert compress_ranges([1, 2, 3, 4, 5, 6]) == '1-6'
- assert compress_ranges([1, 2, 2, 2, 3, 3]) == '1-3'
- assert compress_ranges([1, 1, 2, 2, 3, 3]) == '1-3'
- assert compress_ranges([]) == ''
- assert compress_ranges([1]) == '1-'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 is not correctly compressed to an empty string.

Key Assertions:

- The function should return an empty string when given an empty list as input.
- An empty list should be considered as having no ranges to compress.
- The `compress_ranges` function should handle the case of an empty list without raising an exception or returning a special value.
- The output of the `compress_ranges` function for an empty list should match the expected result (an empty string).
- The test should verify that the function does not produce incorrect results when given an empty input 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: 29-30)

PASSED tests/test_ranges.py::TestCompressRanges::test_mixed_ranges 1ms 3

AI ASSESSMENT

Scenario: Test the function with mixed ranges and singles.

Why Needed: Prevents a bug where the function does not correctly handle mixed ranges and singles.

Key Assertions:

- The function should return the correct compressed range for each single or range.
- The function should correctly group consecutive integers together in the output.
- The function should handle invalid inputs (e.g., negative numbers) without errors.
- The function should preserve the order of the input ranges.
- The function should not add any extra characters to the output (except for commas).
- The function should handle edge cases where there are no consecutive integers in the input.
- The function should correctly compress ranges that span multiple lines of 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/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: Testing the `compress_ranges` function with non-consecutive line numbers.

Why Needed: This test prevents regression where non-consecutive lines are not correctly compressed to a single comma-separated string.

Key Assertions:

- The input list should be sorted in ascending order.
- All elements in the list should be present in the output.
- Non-consecutive line numbers should be separated by commas.
- Any duplicate line numbers should be ignored.
- The function should handle lists with an odd number of elements correctly.
- The function should not throw any errors for invalid input (e.g., negative numbers).
- The function should preserve the original order of non-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: The 'single_line' test verifies that a single-line input does not require range notation.

Why Needed: This test prevents regression where the function compresses an empty list or a list with only one element, which would otherwise incorrectly use range notation.

Key Assertions:

- asserts that the output of `compress_ranges([5])` is '5'.
- asserts that the input `[5]` does not contain any elements to compress.
- asserts that the function correctly handles an empty list.
- asserts that the function correctly handles a list with only one element.
- asserts that the function does not use range notation for single-line inputs.
- asserts that the function returns the expected output '5' when given a single-element input.
- asserts that the function raises an error or returns an appropriate value for empty lists or single-element lists.

COVERAGE

src/pytest_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: The 'test_two_consecutive' test verifies that two consecutive lines in a list of integers use range notation.

Why Needed: This test prevents regression where non-range strings are compressed into single numbers.

Key Assertions:

- assert compress_ranges([1, 2]) == '1-2'
- assert isinstance(compress_ranges([1, 2]), str)
- assert len(compress_ranges([1, 2])) == 3
- assert compress_ranges(['a', 'b']) == 'a-b'
- assert isinstance(compress_ranges(['a', 'b']), str)
- assert len(compress_ranges(['a', 'b'])) == 4
- assert compress_ranges([-1, -2]) == '-1-2'
- assert isinstance(compress_ranges([-1, -2]), str)

COVERAGE

src/pytest_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: The test verifies that the `compress_ranges` function handles an unsorted input correctly.

Why Needed: This test prevents bugs or regressions where the function may incorrectly handle unsorted ranges.

Key Assertions:

- The function should return a string in the format '1-3, 5' for the given input.
- The function should sort the input ranges before returning them.
- The function should not raise an error or throw any exceptions when given unsorted input.
- The function should preserve the original order of equal elements within each range.
- The function should handle duplicate values correctly (if present).
- The function should return a string in the correct format for all possible inputs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 the expand_ranges function might return incorrect results for empty input strings, potentially leading to unexpected behavior or errors in downstream code.

Key Assertions:

- expand_ranges('') returns an empty list
- expand_ranges('abc') returns ['a', 'b', 'c']
- expand_ranges(' def') returns ['d', 'e', 'f']
- expand_ranges('1234567890') returns ['1', '2', '3', ..., '9', '0']
- expand_ranges('abcde') returns ['a', 'b', 'c', 'd', 'e']

COVERAGE

src/pytest_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 a mixed range of values.

Why Needed: This test prevents bugs or regressions where the function incorrectly expands ranges containing both inclusive and exclusive endpoints.

Key Assertions:

- A comma-separated list of ranges is provided as input.
- The function should expand all ranges, including those with inclusive endpoints.
- -1 is not included in the expanded range.
- 10.5 is included in the expanded range.
- 12.3 is excluded from the expanded range.
- -2 is not included in the expanded range.
- 11.7 is included in the expanded range.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 expand a range of numbers into a list.

Why Needed: This test prevents the regression where a single number in a range does not get expanded correctly.

Key Assertions:

- The input string should be '1-3' or any other valid range.
- The function `expand_ranges` should return a list of numbers in the given range.
- The list returned by `expand_ranges` should contain all numbers from the start to the end of the range (inclusive).
- The function should handle ranges with negative numbers correctly.
- The function should handle ranges with zero correctly.
- The function should not throw any errors for invalid input 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/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 the `compress_ranges` and `expand_ranges` functions produce the same output when given a list of numbers.

Why Needed: This test prevents regression where the order of elements in the compressed or expanded ranges is changed unexpectedly.

Key Assertions:

- The original list remains unchanged after compression and expansion.
- The compressed range `[1, 2)` matches the original range `[1, 3]`.
- The expanded range `[1, 5)`, `[10, 15]` matches the original range `[1, 5]` and `[10, 15]` respectively.
- The order of elements in the compressed or expanded ranges is preserved.
- The compressed range `[2, 3)` does not match any other compressed range.
- The expanded range `[11, 12)`, `[15, 16]` matches the original range `[11, 12]` and `[15, 16]` respectively.
- The order of elements in the compressed or expanded ranges is preserved when repeated.
- The function handles empty input lists 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	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 test verifies that the `expand_ranges` function correctly handles a single number input.

Why Needed: This test prevents a regression where the function incorrectly expands numbers to multiple elements.

Key Assertions:

- Input: '5'
- Expected output: [5]
- Function should return a list containing only the single element: 5
- The expansion of the input number should not be affected by other ranges
- No additional elements should be appended to the result
- The function should handle numbers with leading zeros correctly (e.g., '05')
- Numbers outside the valid range should raise an error

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: The test verifies that the `format_duration` function correctly formats durations in milliseconds for values less than 1 second.

Why Needed: This test prevents a regression where the function does not return the expected format for durations greater than or equal to 1 millisecond.

Key Assertions:

- the output is '500ms' when the input is 0.5 seconds.
- the output is '1ms' when the input is 0.001 seconds.
- the output is '0ms' when the input is 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: The function `format_duration` should correctly format the output for durations in seconds.

Why Needed: This test prevents a potential bug where the function does not handle durations greater than or equal to 1 second correctly.

Key Assertions:

- assert format_duration(1.23) == '1.23s'
- assert format_duration(60.0) == '60.00s'

COVERAGE

`src/pytest_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 that all outcomes map to CSS classes correctly.

Why Needed: This test prevents regression of the `outcome_to_css_class` function when it maps non-existent outcomes.

Key Assertions:

- The function returns a CSS class based on the outcome.
- The function handles 'passed' as 'outcome-passed'.
- The function handles 'failed' as 'outcome-failed'.
- The function handles 'skipped' as 'outcome-skipped'.
- The function handles 'xfailed' as 'outcome-xfailed'.
- The function handles 'xpassed' as 'outcome-xpassed'.
- The function returns the correct CSS class for an error 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/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 an unknown outcome is not handled correctly and may result in unexpected styling.

Key Assertions:

- The function `outcome_to_css_class` should return the default class 'outcome-unknown' for an unknown outcome.
- The function `outcome_to_css_class` should raise an exception or handle the situation appropriately when given an unknown outcome.
- The CSS class 'outcome-unknown' should be applied to the element with the unknown 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/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: The test verifies that a basic report is rendered correctly.

Why Needed: This test prevents rendering of an incomplete HTML document, which would display only the 'Test Report' heading and no actual test results.

Key Assertions:

- The presence of '' in the rendered HTML.
- The presence of 'Test Report' in the rendered HTML.
- The presence of 'test::passed' in the rendered HTML.
- The presence of 'test::failed' in the rendered HTML.
- The presence of 'PASSED' and 'FAILED' in the rendered HTML.
- The presence of 'Plugin: v0.1.0' and 'Repo: v1.2.3' in the rendered HTML.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 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: Verifies that the test renders coverage information and includes it in the rendered HTML.

Why Needed: This test prevents a regression where the coverage information is not included in the rendered HTML.

Key Assertions:

- The test should include the 'src/foo.py' file in the rendered HTML.
- The test should display '5 lines' in the rendered HTML.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Test renders LLM annotations for fallback HTML.

Why Needed: This test prevents a potential security vulnerability where an attacker could bypass authentication by manipulating the rendered HTML.

Key Assertions:

- The report includes 'Tests login flow' as part of its content.
- The report includes 'Prevents auth bypass' as part of its content.
- The report contains the specified LLM annotation for the 'Tests login flow' scenario.
- The report displays the specified why_needed message for the 'Prevents auth bypass' scenario.
- The rendered HTML includes the required LLM annotations for the 'Tests login flow' and 'Prevents auth bypass' scenarios.
- The report passes all assertions related to the LLM annotation content.
- The report passes all assertions related to the why_needed message content.
- The report displays the correct key assertions about the rendered HTML.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Verify that the test includes source coverage summary and reports on 'src/foo.py' file.

Why Needed: This test prevents a regression where the source coverage is not reported for files with only one statement.

Key Assertions:

- The report should contain 'Source Coverage' section.
- 'src/foo.py' should be listed in the report.
- The percentage of covered statements should be displayed (80.0%).
- The ranges where missed statements are reported ('5, 9-10') should be included.
- The coverage percent should be calculated correctly (80.0%) and displayed as a percentage.
- 'src/foo.py' file should have exactly 8 covered statements.
- Missed statements range should include '5' and '9-10'.
- Coverage ranges should be formatted correctly ('1-4, 6-8').

COVERAGE

src/pytest_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 summary includes both 'XFailed' and 'XPassed' elements.

Why Needed: This test prevents a regression where the summary is missing or incorrectly formatted for XPass tests.

Key Assertions:

- The HTML string contains the expected 'XFailed' and 'XPassed' text.
- The HTML string does not contain any other text that could interfere with the 'XFailed' and 'XPassed' identification.
- The 'XFailed' and 'XPassed' elements are correctly formatted within the summary section of the rendered report.

COVERAGE

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

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/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 'test_empty_bytes' verifies that an empty bytes object produces consistent hash.

Why Needed: This test prevents a potential issue where the hash of an empty bytes object is not consistent, potentially leading to incorrect reporting or analysis.

Key Assertions:

- The hash of an empty bytes object should be equal to itself (hash1 == hash2)
- The length of the hash of an empty bytes object should be 64 characters (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 duration and version information for a test run.

Why Needed: This test prevents a regression where the build run meta is not correctly calculated, potentially leading to incorrect reporting or analysis of test results.

Key Assertions:

- The duration of the test run should be equal to 60 seconds.
- The `pytest_version` attribute should have a value.
- The `plugin_version` attribute should be set to '0.1.0'.
- The `python_version` attribute should also be present and correctly formatted.

COVERAGE

src/pytest_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 counts all outcome types correctly.

Why Needed: This test prevents a regression where the `build_summary` method incorrectly counts some outcome types as failed or skipped.

Key Assertions:

- total == 6 (all outcomes)
- passed == 1 (only passed outcome)
- failed == 1 (only failed outcome)
- skipped == 1 (only skipped outcome)
- xfailed == 1 (only xfailed outcome)
- xpassed == 1 (only xpassed outcome)
- error == 1 (only error 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/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 the ReportWriter's ability to build a summary with correct counts of passed, failed, skipped and total outcomes.

Why Needed: This test prevents regression where the count of passed, failed or skipped tests is not accurate due to incorrect handling of skipped tests.

Key Assertions:

- The total number of tests should be equal to 4 (2 passed, 1 failed, 1 skipped).
- The number of passed tests should be 2 (tests 'test1' and 'test3').
- The number of failed tests should be 1 (test 'test3').
- The number of skipped tests should be 1 (test 'test4').

COVERAGE

src/pytest_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's configuration is not properly initialized, potentially causing issues with subsequent operations.

Key Assertions:

- The `config` attribute of the `ReportWriter` instance should be set to the provided `Config` object.
- The `warnings` list of the `ReportWriter` instance should be empty.
- The `artifacts` list of the `ReportWriter` instance should be empty.
- The `writer.config` attribute should refer to the original `config` object passed to the `ReportWriter` constructor.
- The `writer.warnings` attribute should be an empty list.
- The `writer.artifacts` attribute should be an empty list.
- If a configuration with warnings is provided, the `writer.warnings` attribute should not contain any warnings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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.

Why Needed: This test prevents regression where the report does not include all tests, potentially leading to missing test results in future reports.

Key Assertions:

- The length of the report.tests list should be equal to 2.
- The total number of tests in the summary.total attribute should be equal to 2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Test 'test_write_report_includes_coverage_percent' verifies that the report includes total coverage percentage.

Why Needed: This test prevents regression where the report does not include the total coverage percentage, which is crucial for ensuring transparency and accuracy in the testing process.

Key Assertions:

- The `coverage_total_percent` attribute of the `report.summary` object should be equal to `85.5` when the `coverage_percent` parameter is set to `85.5`.
- The total coverage percentage calculated by the `writer.write_report()` method should match the expected value of `85.5`.
- The report summary should contain the correct total coverage percentage.

COVERAGE

src/pytest_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: Report should include source coverage summary.

Why Needed: This test prevents a regression where the report does not include source coverage information.

Key Assertions:

- The length of `report.source_coverage` is equal to 1.
- The file path of `report.source_coverage[0]` is exactly 'src/foo.py'.
- All elements in `report.source_coverage` have the correct type (SourceCoverageEntry).

COVERAGE

src/pytest_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 a regression where the report writer does not merge coverage into tests, potentially leading to incorrect reporting of test coverage.

Key Assertions:

- The report should have at least one coverage entry for each test.
- Each coverage entry should be associated with the correct file path.
- All file paths in the first coverage entry should match the expected file 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/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` 6ms 5

AI ASSESSMENT

Scenario: Test that the `ReportWriter` falls back to direct write if an atomic write fails and the file system is unable to perform an atomic write.

Why Needed: This test prevents a regression where the `ReportWriter` may not be able to write reports due to a failure in the underlying operating system or file system.

Key Assertions:

- The file `report.json` should exist after the test.
- At least one warning message should have been generated with code 'W203' during the atomic write 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/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` 6ms 5

AI ASSESSMENT

Scenario: Test verifies that the `ReportWriter` creates an output directory if it does not exist.

Why Needed: This test prevents a bug where the report writer fails to create a directory for the output file when it is missing.

Key Assertions:

- The output directory should be created with the correct name (`subdir/report.json`).
- The output directory should have the correct permissions (in this case, read and write access).
- The `report_json` attribute of the `Config` object should match the expected path.
- The `ReportWriter` instance should be able to create the output directory using its `write_report` method.
- The `tmp_path / subdir / report.json` expression should evaluate to a valid Python path.
- The `exists` method should return True for the created output directory.
- The `writer.write_report(tests)` call should not raise an exception if the output directory does 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	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 prevents the capture of warnings.

Why Needed: To prevent the test from capturing an error message when creating a non-existent directory.

Key Assertions:

- The `writer._ensure_dir(json_path)` function should not raise any exception.
- The `writer.warnings` list should be empty after calling `_ensure_dir(json_path)`.
- Any warnings raised during the creation of the directory should have a code of 'W201'.
- No error messages should be printed to the console when creating the non-existent directory.
- The `Config` object's `report_json` attribute should not be modified by the test.
- The `ReportWriter` object's `_ensure_dir` method should raise an exception with a suitable error message when attempting to create a non-existent directory.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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` values for `sha` and `dirty` when `git not found`,

Why Needed: This test prevents a potential regression where the report writer fails to generate reports due to a failure in the `get_git_info` function.

Key Assertions:

- The `sha` attribute of the returned dictionary is `None`.
- The `dirty` attribute of the returned dictionary is `None`.
- The `git not found` exception is raised when calling `subprocess.check_output` with a side effect.
- The test asserts that the `sha` and `dirty` values are `None` after the test.
- The test does not fail if `git not found` is not raised during the execution of 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/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 verifies that the report writer creates an HTML file with expected content.

Why Needed: This test prevents a regression where the report writer fails to create an HTML file even when there are tests that fail or are skipped.

Key Assertions:

- The report.html file should exist in the same directory as the test.
- The report.html file should contain the expected content (test1, test2, PASSED, FAILED, SKIPPED, XFailed, XPassed, Errors).
- All lines in the report.html file should be present in the correct order (PASSED, FAILED, SKIPPED, XFailed, XPassed, Errors).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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` 32ms 5

AI ASSESSMENT

Scenario: The test verifies that the report includes xfail outcomes in the HTML summary.

Why Needed: This test prevents a regression where the xfail summary is not included in the HTML report.

Key Assertions:

- Asserts that 'XFAILED' and 'XPASSED' are present in the HTML string.
- Asserts that 'XFailed' and 'XPassed' are present in the HTML string.
- Checks if the HTML contains both 'XFAILED' and 'XPASSED'
- Verifies if the HTML includes 'XFailed' but not 'XPassed'
- Ensures the presence of 'XPassed' in the HTML
- Checks for correct case sensitivity in 'XFAILED' and 'XPASSED'
- Verifies if the HTML does not contain any other xfail-related keywords

COVERAGE

src/pytest_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: Test verifies that the `ReportWriter` class creates a JSON file with the specified report.

Why Needed: This test prevents regression where the `ReportWriter` does not create a JSON file for reports.

Key Assertions:

- The file should exist at the path specified by `tmp_path / 'report.json'`.
- The `writer.artifacts` list should contain at least one artifact.
- The `json_path` variable is correctly calculated using `str(tmp_path / 'report.json')`.
- The `config` object created with `Config(report_json=json_path)` has the correct report path.
- The `writer.write_report(tests)` method writes the test results to the JSON file.
- The `writer.artifacts` list is updated correctly after writing the report.
- The number of artifacts in the `writer.artifacts` list is 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` 34ms 5

AI ASSESSMENT

Scenario: Test writes PDF file when Playwright is available.

Why Needed: Prevents regression where Playwright is not available, potentially causing test failures or incorrect report generation.

Key Assertions:

- Verify that a PDF file named 'report.pdf' is created in the specified temporary path.
- Check if any artifact with the name 'report.pdf' exists in the artifacts list of the ReportWriter instance.
- Assert that the file system has been modified to create the 'report.pdf' file at the specified 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/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 that a warning is raised when the Playwright module is missing for PDF output.

Why Needed: To prevent a potential issue where the test fails due to an unhandled `Warning` exception when writing reports with PDF output.

Key Assertions:

- The file 'report.pdf' should not exist.
- Any warning messages raised by the writer should have code `WarningCode.W204_PDF_PLAYWRIGHT_MISSING.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)
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 that the report writer ensures directory creation when a temporary directory is reused.

Why Needed: The test prevents a potential bug where the report writer fails to create a new directory for each report, leading to inconsistent coverage and reports with old or missing files.

Key Assertions:

- Asserts that `tmp_dir.exists()` returns True after the test.
- Asserts that all warnings in `writer.warnings` have the code 'W202' when the temporary directory is reused.
- Asserts that `shutil.rmtree(tmp_dir)` successfully removes the temporary directory 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` 9ms 5

AI ASSESSMENT

Scenario: Test verifies that report_writer_metadata_skips function skips metadata when reports are disabled.

Why Needed: This test prevents a regression where the report writer does not include metadata in its output even when reports are disabled.

Key Assertions:

- The 'start_time' key should be present in the run meta.
- The 'llm_model' key should be None for the run meta.
- The function to_dict() method should return a dictionary with 'start_time' and 'llm_model' keys.

COVERAGE

src/pytest_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` creates a valid annotation from a dictionary with all required fields.

Why Needed: Prevents regression in case of missing or malformed input data, ensuring the correct creation of annotations.

Key Assertions:

- assert schema.scenario == 'Verify login'
- assert schema.why_needed == 'Catch auth bugs'
- assert schema.key_assertions == ['assert 200', 'assert token']
- assert schema.confidence == 0.95

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/schemas.py`	`5 lines (ranges: 77-81)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_schemas.py::TestAnnotationSchema::test_to_dict_full` 1ms 3

AI ASSESSMENT

Scenario: Should convert to dictionary with all fields.

Why Needed: Prevent regression in authentication logic when converting schema to dict.

Key Assertions:

- assert data['scenario'] == 'Verify login', assert data['why_needed'] == 'Catch auth bugs', 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` 84ms 7

AI ASSESSMENT

Scenario: The HTML report is generated correctly and exists.

Why Needed: This test prevents a bug where the report does not exist or has incorrect contents.

Key Assertions:

- The file path of the report should be correct.
- The report should contain the expected content.
- The report name should match the function name.
- The report should have an HTML structure.

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` 119ms 7

AI ASSESSMENT

Scenario: tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_summary_counts_all_statuses

Why Needed: This test prevents regression of HTML summary counts not including 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 'XPassed' label
- assert True is included in the 'Errors' label

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` 72ms 7

AI ASSESSMENT

Scenario: The JSON report is created and its existence and content are verified.

Why Needed: This test prevents a regression where the JSON report is not generated correctly.

Key Assertions:

- The report path exists.
- The report file is in the correct format (JSON).
- The schema version of the report matches the expected value.
- The summary statistics match the expected values (total: 2, passed: 1, failed: 1).

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` 77ms 13

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report when a provider is enabled.

Why Needed: Prevent regressions by ensuring LLM annotations are present in the report.

Key Assertions:

- The scenario 'Checks the happy path' should be reported in the LLM annotation.
- The reason 'Prevents regressions' should be included in the LLM annotation.
- The key assertions 'asserts True' should be included in the LLM annotation.
- The expected value of the 'scenario' field in the LLM annotation is correct.
- The expected value of the 'why_needed' field in the LLM annotation is correct.
- The expected value of the 'key_assertions' field in the LLM annotation is correct.
- The expected values of the 'choices' field in the LLM annotation are correct.

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 that the report contains 'LLM error' and 'boom'.
- The test asserts that both 'LLM error' and 'boom' are present in the report content.
- The test checks for the presence of LLM errors in the report output.
- The test ensures that LLM errors are reported correctly using the provided model.
- The test verifies that the report is generated with the correct model.
- The test checks if the error message contains 'boom' as expected.
- The test confirms that the error message does not contain any other relevant information.

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 57ms 7

AI ASSESSMENT

Scenario: Verify that the LLM opt-out marker records itself in the report.

Why Needed: Prevent a regression where the LLM opt-out marker does not record its presence in the test report.

Key Assertions:

- The 'llm_opt_out' marker is present in the test report.
- The 'llm_opt_out' marker is set to True for this test.
- The number of tests with the 'llm_opt_out' marker is exactly 1.
- The first test with the 'llm_opt_out' marker has its 'llm_opt_out' attribute set to True.

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 56ms 7

AI ASSESSMENT

Scenario: The test verifies that the requirement marker is correctly recorded on a test file.

Why Needed: This test prevents a potential bug where the requirement marker is not properly recorded, potentially leading to incorrect reporting of tests.

Key Assertions:

- The 'requirement' keyword is used with two different requirements (REQ-001 and REQ-002) in the same test file.
- The test function `test_with_req()` is defined without any additional requirements.
- The `pytest.mark.requirement` decorator is applied to both `test_with_req()` and another test function.
- The `requirements` attribute of each test function is not provided or empty.
- The 'REQ-001' and 'REQ-002' requirements are not properly separated in the test file.
- The test data is not correctly loaded from the report.json file.
- The tests list only contains one test, but it should contain multiple tests for a requirement marker to be effective.

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` 62ms 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:

- 2
- ['xfailed', 'xfailed']

COVERAGE

src/pytest_llm_report/collector.py	47 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-116, 119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212, 216, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269,

	276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 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 56ms 7

AI ASSESSMENT

Scenario: The test verifies that the 'test_skip' marker is applied to a function and its documentation.

Why Needed: This test prevents regression by ensuring that the 'test_skip' marker is correctly applied when running tests with the '--llm-report-json' flag.

Key Assertions:

- The 'test_skip' marker should be present in the test function's docstring.
- The 'test_skip' marker should be present in the test function's documentation.
- The number of skipped tests should match the expected value (1 in this case).
- The 'summary' key in the report data should contain a single integer value representing the number of skipped tests.
- The 'skipped' key in the report data should contain an integer value equal to 1.
- The test function's docstring should include the 'test skip' marker with the correct reason.
- The test function's documentation should include the 'test skip' marker with the correct reason.
- The number of skipped tests in the report data should match the expected value (1 in this case).

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` 58ms 7

AI ASSESSMENT

Scenario: Verifies that the 'xfail' marker correctly identifies Xfailed tests in the test report.

Why Needed: This test prevents regression by ensuring that Xfailed tests are recorded and reported accurately.

Key Assertions:

- The number of Xfailed tests is correctly identified in the test report.
- The 'summary' key under 'xfailed' contains a value equal to 1 for each Xfailed test.
- The 'xfail' marker is applied to at least one test function (in this case, `test_xfail`)
- The test report includes the correct count of Xfailed tests (1 in this case).
- The 'summary' key under 'xfailed' contains a value that matches the expected number of Xfailed tests.
- Each Xfailed test is correctly identified and recorded in the test report.

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` 62ms 7

AI ASSESSMENT

Scenario: Test parameterized tests are recorded separately.

Why Needed: This test prevents regression in parametrized tests by ensuring that the correct number of tests are run for each input value.

Key Assertions:

- The total number of tests is 3 (1 passed, 2 failed).
- Each input value x has a corresponding test in the report.
- All tests have been executed successfully.

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` 50ms 3

AI ASSESSMENT

Scenario: The test verifies that the CLI help text contains usage examples.

Why Needed: This test prevents a potential bug where users are not informed about available usage examples in the CLI help text.

Key Assertions:

- assert 'Example:*--llm-report*' in result.stdout
- assert 'Usage: lllm --help' in result.stdout
- assert 'LLM Report:' in result.stdout

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	45 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	118 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 387-388, 391, 395-397)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_markers_registered` 45ms 3

AI ASSESSMENT

Scenario: Verify that LLM markers are registered and their usage is detected by pytester.

Why Needed: This test prevents a potential bug where the LLM markers are not properly registered, causing pytest to fail or behave unexpectedly when running tests with these markers.

Key Assertions:

- The 'llm_opt_out' marker should be found in the output of `pytest --markers`.
- The 'llm_context' marker should be found in the output of `pytest --markers`.
- The 'requirement' marker should be found in the output of `pytest --markers`.
- The LLM markers should not be removed or hidden from the test report.
- The LLM markers should be detected by pytester and trigger the correct tests.
- The LLM context marker should be detected as a separate entity from other context markers.
- The requirement marker should be detected as a separate entity from other requirements.

COVERAGE

src/pytest_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` 52ms 3

AI ASSESSMENT

Scenario: The plugin is successfully registered and its help message can be displayed.

Why Needed: This test prevents a potential issue where the plugin's registration fails or does not display the expected help message.

Key Assertions:

- pytester.runpytest('--help') should return an empty string if no errors are found.
- result.stdout.fnmatch_lines(['*--llm-report*']) should contain '--llm-report' in the output.
- result.stdout.fnmatch_lines(['--help']) should not contain any other lines than '--llm-report'.
- pytester.runpytest('--help') should display the help message without any errors or warnings.
- result.stdout.is_empty() should be True if no errors are found and the help message is displayed correctly.
- result.stdout.fnmatch_lines(['*--llm-report*', '--help']) should contain '--llm-report' in the output and not other lines.
- pytester.runpytest('--help') should display the help message without any additional lines or warnings.
- result.stdout.is_empty() should be True if no errors are found and the help message is displayed correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 84ms 7

AI ASSESSMENT

Scenario: Test verifies that special characters in nodeid are handled correctly by Pytest.

Why Needed: This test prevents a regression where special characters in nodeids cause the application to crash or produce invalid HTML reports.

Key Assertions:

- The string 'hello' is successfully parsed and does not contain any special characters in the nodeid.
- The string 'foo&bar' is successfully parsed and does not contain any special characters in the nodeid.
- The report.html file exists after running the test with --llm-report
- The HTML content of the report.html file contains ''
- The nodeids are correctly formatted without any special characters

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 'format_duration' function with a duration of exactly one minute.

Why Needed: Prevents regression in formatting durations to exactly one minute.

Key Assertions:

- The result should be '1m 00.0s'.
- The format string should include 'm' for minutes and 's' for seconds.
- The function should handle cases where the input is less than or equal to zero.
- The function should handle cases where the input is exactly one minute (60 seconds).
- The function should handle cases where the input is a negative number.
- The function should return an error message when given a non-numeric input.
- The function should correctly format durations in the 12-hour clock format.
- The function should not silently truncate the result for very large inputs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Test formats sub-millisecond durations as microseconds.

Why Needed: Prevents incorrect formatting of microsecond durations.

Key Assertions:

- The function `format_duration(0.0005)` should return '500µs' when called with a duration of 0.0005 seconds.
- The string 'µs' is present in the result of `format_duration(0.0005)`.
- The value returned by `format_duration(0.0005)` is 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: Tests the `format_duration` function to ensure it correctly formats sub-second durations as milliseconds.

Why Needed: This test prevents a potential bug where the function does not format durations with milliseconds when they are less than one second.

Key Assertions:

- The output of `format_duration(0.5)` should contain 'ms' in its string representation.
- The value of `result` should be equal to '500.0ms'.
- When the input duration is exactly one second, the function should return a string with two decimal places (e.g., '500.00ms').

COVERAGE

src/pytest_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: Test the 'minutes' key assertion for durations over a minute.

Why Needed: This test prevents regression when durations are formatted to show minutes and seconds.

Key Assertions:

- The result should contain the string 'm' (minutes) followed by an optional unit ('s').
- The duration value should be in the format '1m 30.5s'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 scenario where it formats multiple minutes.

Why Needed: This test prevents regression in formatting times longer than one minute.

Key Assertions:

- The output should be in the format 'mm:ss'.
- The seconds part should be a non-negative number.
- The minutes part should have leading zeros if necessary.
- The function should handle cases where the input is zero or negative.
- The function should return an error for invalid inputs (e.g., non-numeric values).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 duration of exactly one second.

Why Needed: This test prevents a potential bug where the function returns an incorrect string representation for durations less than or equal to one second.

Key Assertions:

- The result should be '1.00s' when the input is 1.0.
- The result should be '1.00' when the input is exactly 1.0 and no fractional seconds are present.
- No fractional seconds should be included in the output string for durations less than or equal to one 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/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED tests/test_time.py::TestFormatDuration::test_seconds_format 1ms 3

AI ASSESSMENT

Scenario: Verifies that the `format_duration` function correctly formats seconds under a minute.

Why Needed: Prevents incorrect formatting of seconds, where they are displayed as 's' instead of 's.0' or '1.s' for one-second and two-second intervals respectively.

Key Assertions:

- The result contains the string 's', which indicates that the function correctly handles seconds under a minute.
- The result is equal to '5.50s', demonstrating that the function formats seconds accurately.
- The test asserts that the `format_duration` function returns a string containing the correct unit ('s') for 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	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 small millisecond duration.

Why Needed: This test prevents regression in handling durations smaller than 1 millisecond, where the default format is '1s'.

Key Assertions:

- The result of `format_duration(0.001)` should be '1.0ms'.
- The function should correctly handle durations smaller than 1 millisecond.
- The unit 'ms' should be displayed instead of 's' when the duration is small.
- The function should preserve the original value of the input duration.
- The test should fail if the input duration is not a number or is less than 0.
- The test should pass if the input duration can be converted to seconds and milliseconds correctly.
- The format string '1.0ms' should be displayed when the function returns '1.0ms'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 function correctly formats very small durations as microseconds.

Why Needed: This test prevents a potential bug where the function may incorrectly format very small durations as seconds or milliseconds instead of microseconds.

Key Assertions:

- The result should be '1µs' when the input is 0.000001 seconds.
- The result should not exceed the maximum allowed value for microseconds (1023).
- The function should handle negative numbers correctly and return '1ms' instead of '-1ms'.

COVERAGE

src/pytest_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: Tests the function `iso_format` with a datetime object representing UTC time.

Why Needed: Prevents regression in handling datetime objects with UTC timezone.

Key Assertions:

- The output of `iso_format(dt)` should be '2024-01-15T10:30:45+00:00'.
- The `tzinfo` parameter is set to `UTC` for the datetime object.
- The function correctly formats the datetime with UTC timezone.
- The formatted string does not include any time zone offset or offset sign.
- The output does not contain any extra characters that might indicate an incorrect format.
- The function handles datetime objects from different time zones correctly.
- The `iso_format` function is able to convert a datetime object with UTC timezone into the expected ISO format 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/time.py	1 lines (ranges: 27)

PASSED tests/test_time.py::TestIsoFormat::test_formats_naive_datetime 1ms 3

AI ASSESSMENT

Scenario: Verifies that naive datetime is formatted correctly without timezone.

Why Needed: Prevents a potential bug where the naive datetime format is incorrect due to missing timezone information.

Key Assertions:

- The output of `iso_format(dt)` should be '2024-06-20T14:00:00' as expected.
- The input `dt` should be a naive datetime object without timezone information.
- The resulting string should not contain any timezone information.
- Any timezone-related errors or exceptions should be raised for invalid inputs.
- The output format should match the expected ISO 8601 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	1 lines (ranges: 27)

PASSED tests/test_time.py::TestIsoFormat::test_formats_with_microseconds 1ms 3

AI ASSESSMENT

Scenario: Tests the `iso_format` function with a datetime object containing microseconds.

Why Needed: Prevents a potential bug where microseconds are not correctly formatted as ISO 8601.

Key Assertions:

- The output of `iso_format(dt)` should contain the string '123456'.
- The string '123456' is present in the output.
- The value of `dt.microsecond` is greater than or equal to 0.
- The value of `dt.microsecond` is less than 1000000 (1 million).
- The value of `dt.microsecond` is not a multiple of 1000 (1,000).

COVERAGE

src/pytest_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 test cases where the `datetime` module's default timezone is not set to UTC.

Key Assertions:

- The returned datetime object has a non-null tzinfo attribute.
- The returned datetime object has a tzinfo attribute equal to `UTC`.
- A datetime object with an associated UTC timezone is returned.

COVERAGE

src/pytest_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 that the `utc_now()` function returns a current time within a specified tolerance.

Why Needed: This test prevents potential issues where the `utc_now()` function is not returning the correct current time due to timing differences between the UTC timezone and the system's local timezone.

Key Assertions:

- The result of `utc_now()` should be a date and time that is within a tolerance of the original `before` value (<= `result` <= `after`).

COVERAGE

src/pytest_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 `utc_now()` function should return a datetime object.

Why Needed: This test prevents a potential bug where the function returns an incorrect type (datetime) instead of a datetime object.

Key Assertions:

- result is an instance of `datetime`
- result has a valid `date` attribute
- result has a valid `time` attribute
- result is not None
- result is a subclass of `datetime`
- result's `now()` method returns the correct datetime value
- result's `now()` method does not raise an exception if invalid input

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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,	66, 90-91, 192, 203

					196, 205, 217, 219-233, 235, 237, 245-246, 248-249, 251, 253-255, 259, 262-263, 265-266, 269-271, 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	-