Test Report

Run ID: 20979465813-py3.12 • Generated: 2026-01-14 01:56:16 • Duration: 34.56s
Plugin: v0.1.0 (2f498263985a34902252c53c11fb820445bd8f21) [dirty]
Repo: v0.1.0 (dd1fc9a372e8de042d250877a534f1f3d8b460de)
LLM: ollama / llama3.2:1b (minimal context, 381 annotated, 5 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 the aggregate function correctly handles all policy when aggregating multiple test cases.

Why Needed: This test prevents regression in case the 'all' policy is used for multiple test cases, where only one test case was aggregated.

Key Assertions:

- The number of tests retained after aggregation should be equal to the total number of test cases.
- Each report should contain at least two tests.
- All reports should have the same retention count.
- The aggregate function should correctly handle all policy when aggregating multiple test cases.
- The 'all' policy should not affect the retention count of individual test cases.
- The aggregate function should preserve the original order of test cases in each report.
- The aggregate function should preserve the original duration of test cases in each report.
- The aggregate function should preserve the original phase of test cases in each report.

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: The aggregate function should not be called when the aggregation directory does not exist.

Why Needed: This test prevents a potential bug where the aggregate function fails to execute due to an incorrect assumption about the existence of the aggregation directory.

Key Assertions:

- The `aggregator.aggregate()` method is called with no arguments.
- A `Path` object representing the aggregation directory does not exist.
- No exception is raised when calling `aggregator.aggregate()`.
- The aggregate function returns a value (None in this case).

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 `aggregate` function picks the latest policy for a given test case.

Why Needed: This test prevents regression where the `aggregate` function incorrectly selects an older policy when multiple runs of the same test are performed at different times.

Key Assertions:

- The test case should be marked as passed in the result.
- The number of tests in the result should be 1.
- The outcome of the first test in the result should be 'passed'.
- The `run_meta` object should have a `run_count` attribute equal to 2.
- The `summary` object should have a `passed` attribute equal to 1 and a `failed` attribute equal to 0.
- The `aggregate` function should correctly select the latest policy for a given test case.

COVERAGE

src/pytest_llm_report/aggregation.py	77 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 130-131, 134, 141, 146, 148-153, 155, 157-159, 170, 182, 184-188, 190-191, 194, 217, 219-223, 235, 245, 248-249, 251, 253, 275-278, 280)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The aggregator function should raise an error when no directory configuration is provided.

Why Needed: To prevent a potential bug where the aggregator function attempts to aggregate data without a specified directory configuration.

Key Assertions:

- mock_config.aggregate_dir is set to None before calling agg.aggregate()
- agg.aggregate() returns None, indicating that no aggregation was performed
- The aggregator function does not raise an error when called with a mock_config that has an empty aggregate_dir attribute

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: Test that aggregate function returns None when no reports exist or are not found.

Why Needed: To prevent a potential AttributeError in the aggregate function, where it attempts to access 'reports' attribute of an empty list.

Key Assertions:

- The aggregate function should return None for scenarios where 'aggregator.aggregate()' is called with an empty list.
- The aggregate function should not attempt to access 'reports' attribute of an empty list.
- A TypeError should not be raised when calling 'aggregator.aggregate()'.

COVERAGE

`src/pytest_llm_report/aggregation.py`	`9 lines (ranges: 52, 55-57, 109-110, 113-114, 170)`
`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

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

AI ASSESSMENT

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

Why Needed: Prevents regression in core functionality due to changes in aggregation logic.

Key Assertions:

- coverage was correctly deserialized with the expected file paths and line ranges
- LLM annotation was correctly deserialized with the expected scenario, why needed, and key assertions
- Can be re-serialized successfully after fix

COVERAGE

src/pytest_llm_report/aggregation.py	81 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 130-131, 134-137, 141-144, 146, 148-153, 155, 157-159, 170, 182, 184-188, 194, 217, 219-223, 235, 245, 248-249, 251, 253, 275-278, 280)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	32 lines (ranges: 40-43, 104-107, 109-111, 113, 115, 161-165, 167, 169, 171, 173, 176-180, 182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that source coverage summary is deserialized correctly.

Why Needed: This test prevents a bug where the source coverage summary is not properly deserialized from the aggregated report.

Key Assertions:

- The 'source_coverage' key in the aggregated report should be an array of SourceCoverageEntry objects.
- Each SourceCoverageEntry object should have the following keys: 'file_path', 'statements', 'missed', 'covered', 'coverage_percent', 'covered_ranges', and 'missed_ranges'.
- The 'source_coverage' value of each SourceCoverageEntry object should be an integer representing the number of statements missed.
- The 'covered' value of each SourceCoverageEntry object should be an integer representing the number of statements covered.
- The 'coverage_percent' value of each SourceCoverageEntry object should be a float between 0 and 1 representing the percentage of covered code.
- The 'covered_ranges' value of each SourceCoverageEntry object should be a string representing the ranges of lines that were not covered.
- The 'missed_ranges' value of each SourceCoverageEntry object should be a string representing the ranges of lines that were missed.
- Each SourceCoverageEntry object should have an 'aggregate' method that returns the aggregated coverage data for the source file.

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 a potential bug where the test fails due to an unconfigured or non-existent source file.

Key Assertions:

- Verify that `_load_coverage_from_source()` returns `None` when `llm_coverage_source` is `None`.
- Verify that `_load_coverage_from_source()` raises a `UserWarning` with message 'Coverage source not found' when the file does not exist.
- Verify that `_load_coverage_from_source()` correctly loads coverage data from a mock `SourceCoverageEntry` instance.
- Verify that `_load_coverage_from_source()` calls `coverage.Coverage.load()` once to load the coverage data.
- Verify that `_load_coverage_from_source()` calls `pytest_llm_report.coverage_map.CoverageMapper.map_source_coverage()` with the mock `CoverageMapper` instance.
- Verify that the returned result is not `None` and contains exactly one entry.
- Verify that each entry in the result has the correct properties (e.g., `file_path`, `statements`, etc.).

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 aggregator recalculates summary for various test cases.

Why Needed: This test prevents regression in the aggregation process when encountering tests with varying durations and outcomes.

Key Assertions:

- The total count of tests passed should be equal to the number of tests provided.
- The number of failed tests should be equal to the number of tests provided.
- The number of skipped tests should be equal to the number of tests provided.
- The number of tests with 'xfailed' outcome should be equal to the number of tests provided.
- The total duration of all tests should be preserved after aggregation.
- The coverage percentage should remain unchanged after aggregation.

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 that skipping an invalid JSON file prevents a regression.

Why Needed: This test verifies that the `aggregate` function correctly handles missing or invalid JSON files, preventing a potential bug where it incorrectly counts such files as skipped.

Key Assertions:

- The aggregate function should not count any reports with missing fields.
- The aggregate function should only count valid reports (i.e., those with all required fields).
- The aggregate function should raise a `UserWarning` when encountering an invalid JSON file, indicating that it was skipped.
- The aggregate function should ignore the invalid report and continue counting valid reports.
- The aggregate function should not incorrectly increment the run meta run count for the invalid report.
- The aggregate function should only consider the first valid report in a directory as counted.
- The aggregate function should raise an error when encountering multiple invalid JSON files in a directory.

COVERAGE

src/pytest_llm_report/aggregation.py	71 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119-120, 125, 127-128, 148-153, 155, 157-159, 162, 164-166, 168, 170, 182, 184-186, 194, 217, 219-220, 235, 245, 248-249, 251, 253, 275-278, 280)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the aggregator recalculates the summary correctly with a given set of tests and latest summary.

Why Needed: This test prevents regression if the aggregator's `_recalculate_summary` method changes without updating the coverage_total_percent in the Summary object.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/aggregation.py	10 lines (ranges: 44, 217, 219-225, 235)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression in cases where caching is enabled and the annotator is not properly configured to handle it.

Key Assertions:

- mock_provider should be mocked with a non-annotated provider
- mock_cache should be mocked with an empty cache
- mock_assembler should be mocked with no assembler
- the annotator's caching mechanism should be tested and validated
- the annotator's caching mechanism should not interfere with the test execution

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-81, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that annotators can be concurrently annotated without causing any issues.

Why Needed: This test prevents a potential issue where multiple annotators are trying to annotate the same document at the same time, leading to inconsistent results or errors.

Key Assertions:

- Ensures that all annotations are processed sequentially and in the correct order.
- Verifies that no annotation is skipped or lost due to concurrent processing.
- Checks for any potential data corruption or inconsistencies caused by concurrent access to the document.
- Ensures that annotators can handle large documents without experiencing performance issues.
- Verifies that the annotator's cache is properly updated and synchronized after concurrent annotations.
- Checks for any unexpected behavior when multiple annotators are trying to annotate the same document simultaneously.

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 concurrently without any noticeable impact on the annotation process.

Why Needed: This test prevents a potential regression where concurrent annotations cause failures and the annotator crashes or freezes.

Key Assertions:

- Verify that the annotator does not crash or freeze when handling multiple failures concurrently.
- Check if the annotator can still annotate successfully even when there are multiple failures in the queue.
- Verify that the annotator correctly handles failed annotations and continues with the next task.
- Test that the annotator does not introduce any significant performance degradation due to concurrent annotation.
- Check if the annotator's error handling mechanism is able to recover from a failure without causing the test to fail.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104-112, 129-135, 137-139, 229-232, 234, 236-237, 239, 245-246, 248-253, 255, 261-264, 266)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The `test_progress_reporting` function is called with mock providers and cache to test progress reporting.

Why Needed: This test prevents regression in the annotator's ability to report progress accurately when using mock data.

Key Assertions:

- mock_provider.get() should return a mock object that can be mocked again.
- mock_cache.get() should return None for non-existent keys.
- mock_assembler.get() should not throw an exception if the key is not found.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verifies the sequential annotation functionality of the annotator.

Why Needed: This test prevents regression in sequential annotation where multiple annotations are performed sequentially without any intermediate steps.

Key Assertions:

- The function `test_sequential_annotation` should not throw an exception when called with mock providers, mock caches and mock assemblers.
- The annotator should correctly annotate the input data for each step of the sequential annotation process.
- There should be no intermediate errors or exceptions thrown during the sequential annotation process.
- The output annotations from each step should match the expected output.
- The annotator's cache should not be modified unexpectedly by the sequential annotation process.
- Each mock provider, mock cache and mock assembler should have a unique name.
- The function `test_sequential_annotation` should correctly handle nested annotations.

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: Verify that the annotator does not skip tests when LLM is disabled.

Why Needed: Prevent regression in case the LLM is disabled and the test is run without it.

Key Assertions:

- The `Config` object with provider='none' should be created successfully.
- An empty list `[]` should be passed to the `annotate_tests` function.
- The `LLM` annotation should not be skipped when `config.llm_enabled` is False.
- No error message or exception should be raised when `config.llm_enabled` is False.
- The annotator's behavior should remain unchanged when LLM is disabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 test verifies that the annotator skips annotation if the provider is unavailable.

Why Needed: This test prevents a potential regression where the annotator fails to skip annotation when the provider is not available.

Key Assertions:

- mock_provider.is_unavailable()
- mock_provider.skip_annotation()
- self.assertEqual(mock_provider.is_skipped(), True)
- self.assertTrue(self.mocked_capsys.readout())
- mock_provider.is_skipped() == mock_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_annotator_maximal.py::TestAnnotatorAdvanced::test_annotate_concurrent_with_progress_and_errors` 2ms 4

AI ASSESSMENT

Scenario: Test the annotator's behavior when annotated concurrently with progress and errors.

Why Needed: This test prevents a potential regression where the annotator fails to report progress or first error in concurrent mode.

Key Assertions:

- Verify that the annotator reports both 'first error' and 'Processing X test(s)' messages as expected.
- Confirm that at least one annotation task has an LLMAnnotation with the key 'error'.
- Check if the annotator correctly identifies the first error in the list of annotations.
- Ensure that the progress_msgs contain the expected strings.
- Verify that the annotator reports progress for all tasks.
- Test that the annotator correctly handles multiple annotation tasks concurrently.

COVERAGE

src/pytest_llm_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: Prevents regression where the annotator does not wait for the rate limit interval to elapse before proceeding with sequential annotation.

Key Assertions:

- assert mock_sleep.called
- assert mock_time.call_count == 5
- assert mock_time.side_effect == [100.0, 100.1, 100.2, 100.3, 100.4]
- assert tasks[0].outcome == 'p' and tasks[1].outcome == 'p'
- assert tasks[0].nodeid == 't1' and tasks[1].nodeid == 't2'
- assert tasks[-1].outcome == 'p'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	23 lines (ranges: 165-168, 170-171, 173-174, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `progress_msgs` list should contain any messages indicating that a test was annotated from a cache.
- Each message in `progress_msgs` should start with '(cache): ' to indicate that the annotation came from a cache.
- If no cached tests were annotated, all messages in `progress_msgs` should be empty.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: The test verifies that when the provider is not available, it prints a message and returns without attempting to annotate tests.

Why Needed: This test prevents a regression where the annotator fails to report annotation failures due to an unavailable provider.

Key Assertions:

- mocks.get_provider().is_available.return_value == False
- assert captured.out.contains('not available. Skipping annotations')
- mocks.get_provider().is_available.assert_called_once_with()
- mocks.get_provider().is_available.return_value.call_count == 1
- mocks.get_provider().get_provider() is None
- mocks.get_provider().is_available.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/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that a `MockProvider` instance correctly raises an error when attempting to parse a malformed JSON response after extracting the JSON.

Why Needed: This test prevents a bug where the `_parse_response` method of `MockProvider` fails with a `JSONDecodeError` due to invalid JSON in the extracted response.

Key Assertions:

- The expected error message is 'Failed to parse LLM response as JSON'.
- The `annotation.error` attribute matches this expected error message.
- The `provider._parse_response(response)` call returns an instance of `MockProviderError` with the specified error message.
- The `MockProviderConfig()` instance passed to `MockProvider` is not used in the test.
- The JSON string provided as input to `_parse_response` has invalid syntax.
- The extracted JSON from the response does not match the expected format.
- The `extract_json_from_response` method finds braces, but the contents 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/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: Test that the `test_base_parse_response_non_string_fields` function correctly handles non-string fields in the response data.

Why Needed: This test prevents a potential bug where the function does not handle non-string fields correctly, leading to incorrect parsing or errors.

Key Assertions:

- The correct scenario is set when the `response_data` dictionary contains an integer value.
- The correct why_needed list includes 'list' as one of the expected reasons for failure.
- The correct key_assertion is included in the assertion for the `annotation` object, which represents the parsed response 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	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: Verifies that the `get_gemini_provider` function returns a valid instance of `GeminiProvider`.

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

Key Assertions:

- The returned value is an instance of `GeminiProvider`.
- The returned value has the correct class name (`GeminiProvider`).
- The returned value does not have any additional attributes or methods (e.g., no `__init__`, `name`, 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	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: Verify that a ValueError is raised when an unknown LLM provider is specified in the configuration.

Why Needed: This test prevents regression where an invalid provider is used without raising an error.

Key Assertions:

- The function `get_provider` raises a ValueError with message 'Unknown LLM provider: invalid' when called with a config object of type `Config` that has a provider set to 'invalid'.
- The test fails if the `test_get_invalid_provider` method is called with a valid configuration object.
- The test passes if the `get_provider` function correctly raises a ValueError for an unknown provider.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 245, 247, 249, 252, 257, 262, 267)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: Prevents regression when switching to litellm provider for testing purposes.

Key Assertions:

- The returned object should be an instance of `LiteLLMProvider`.
- The returned object should have the correct attributes (e.g. name, description).
- The returned object should not raise any exceptions during instantiation.
- The returned object should implement the necessary interfaces (e.g. `get_model`, `get_session`).
- The returned object's provider attribute should be set to 'litellm'.
- The returned object's name and description attributes should match the expected values.
- The returned object's model and session attributes should have the correct types 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/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 a NoopProvider is returned when no provider is specified in the configuration.

Why Needed: Prevents a potential bug where the test fails due to an incorrect assumption about the behavior of get_provider() when no provider is provided.

Key Assertions:

- The function get_provider() returns a NoopProvider instance.
- The function get_provider() returns None if no provider is specified in the configuration.
- The assert statement passes if the returned provider is indeed a NoopProvider instance.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 245, 247, 249-250)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the provider is not correctly initialized with the 'ollama' provider.

Key Assertions:

- The `provider` attribute of the returned `OllamaProvider` instance should be an instance of `OllamaProvider`.
- The `provider` attribute of the returned `OllamaProvider` instance should not be `None`.
- The `provider` attribute of the returned `OllamaProvider` instance should have a valid `name` attribute.
- The `provider` attribute of the returned `OllamaProvider` instance should have a valid `maximal` attribute.
- The `provider` attribute of the returned `OllamaProvider` instance should be an instance of `OllamaProvider` with a valid `config` object.
- The `provider` attribute of the returned `OllamaProvider` instance should not be an instance of any other provider 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	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 the LlmProvider Defaults class returns a provider with available caches.

Why Needed: This test prevents a regression where the LlmProviderDefaults class does not return an instance of LlmProvider when it is expected to have available caches.

Key Assertions:

- The provider should be `is_available()`
- The provider should be `is_available()` again after calling `_check_availability`
- The number of checks performed by the provider should increase by 1 each time it is called `_check_availability`
- The provider's internal state (number of checks) should remain consistent across multiple calls to `_check_availability`

COVERAGE

src/pytest_llm_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: The 'get_model_name()' method of the 'ConcreteProvider' class should return the default model name from the provided configuration.

Why Needed: This test prevents a potential bug where the default model name is not correctly retrieved from the configuration, potentially leading to incorrect model names being used in downstream applications.

Key Assertions:

- The method `get_model_name()` of the class `ConcreteProvider` should return 'test-model' if no other provider is specified.
- The method `get_model_name()` of the class `ConcreteProvider` should return 'test-model' if a model name is provided in the configuration.
- The method `get_model_name()` of the class `ConcreteProvider` should raise an exception with a meaningful error message when the default model name cannot be determined 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/llm/base.py	3 lines (ranges: 52-53, 136)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verifies that the `get_rate_limits` method of a `ConcreteProvider` instance returns `None` when no rate limits are specified in the configuration.

Why Needed: This test prevents a potential bug where the default rate limit for LLM providers is not properly set to None when no custom rate limits are provided.

Key Assertions:

- The `get_rate_limits()` method of the `ConcreteProvider` instance returns `None`.
- No rate limits are specified in the configuration.
- The default rate limit for LLM providers is not properly set to `None` when no custom rate limits are provided.
- The `get_rate_limits()` method does not throw an exception or raise an error when called with no rate limits.
- The `get_rate_limits()` method returns a value that is not `None` and can be used for further processing.
- The `get_rate_limits()` method should return `None` instead of some other value when no rate limits are specified.
- The configuration object passed to the `ConcreteProvider` instance does not contain any rate limit settings.
- The provider instance is created with a valid configuration that includes rate limits, but the `get_rate_limits()` method still returns `None`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 128)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 value for `is_local` is changed to False.

Key Assertions:

- the `provider.is_local()` method should return `False` when the default value is `False`
- the `provider.is_local()` method should not be affected by external changes to the default 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	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: Testing the consistency of a cache with a single source function.

Why Needed: This test prevents a potential bug where different sources produce different hashes, potentially leading to inconsistent caching behavior.

Key Assertions:

- The `hash_source` function should return the same hash for the given source function.
- The `hash_source` function should raise an error if the source function is not found.
- The `hash_source` function should use the original source code as input, rather than a compiled version or string representation.

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 code snippets.

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

Key Assertions:

- The `hash_source` function should return a non-zero hash value when given two different source code strings.
- The `hash_source` function should not produce the same hash for two different source code strings.
- The `hash_source` function should raise an exception or return None when given invalid input (e.g., empty string).
- The `hash_source` function should preserve the original source code format and syntax when generating a unique hash.
- The `hash_source` function should not generate a hash for a single-line comment or other non-code snippet.
- The `hash_source` function should raise an exception if given a string that is not a valid Python source code.
- The `hash_source` function should return the same hash value when given two different strings with the same content but different order of statements.
- The `hash_source` function should preserve the original line numbers and indentation when generating a unique hash.

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 hash generated by `hash_source`.

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

Key Assertions:

- The hash should be at least 16 characters long.
- The hash should be no longer than 32 characters long.
- The hash should have a consistent length for all inputs.
- The hash should not change when the input string is modified or updated.
- The hash should not produce a hash that is too short to be considered valid.
- The hash should not produce a hash that is too long to be considered valid.
- The hash should have a consistent length for different inputs with the same characters.
- The hash should have a consistent length for different inputs with different characters.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that clearing the cache removes all entries.

Why Needed: Prevents regression where cache entries are not cleared after test execution.

Key Assertions:

- The `clear` method should remove all cache entries.
- The `get` method for a non-existent key should return `None`.
- All cache entries should be removed from the cache.
- No exceptions should be raised when clearing the cache.
- The number of cache entries should be equal to 2 after clearing.
- The values associated with the test::a and test::b keys should be None.

COVERAGE

src/pytest_llm_report/cache.py	26 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 129, 132-136, 141)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: To prevent caching of error-related annotations, ensuring that these are retrieved from the database instead.

Key Assertions:

- The annotation 'test::foo' should be present in the cache without a value.
- The annotation 'test::foo' should have an error message ('API timeout') when retrieved from the cache.
- The annotation 'test::foo' should not be cached with a value of 'abc123'.

COVERAGE

src/pytest_llm_report/cache.py	11 lines (ranges: 39-41, 53, 55-56, 86, 88, 118-119, 121)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where an entry with no corresponding key is not correctly returned from the `get` method.

Key Assertions:

- The function should return `None` when the provided key does not exist in the cache.
- The function should raise a `KeyError` exception if the provided key does not exist in the cache.
- The function should not throw an exception if the provided key exists in the cache but its value is empty or None.
- The function should correctly handle missing keys with no corresponding values.
- The function should correctly handle missing keys with corresponding values (e.g., `test::foo` with a non-empty string)
- The function should not return an exception when the key does not exist in the cache but its value is empty
- The function should not throw an exception when the key exists in the cache but its value is None

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 setting and getting annotations from the cache.

Why Needed: Prevents bypass by ensuring that annotations are stored in the cache before being used.

Key Assertions:

- Check if the annotation is set correctly in the cache.
- Verify that the annotation's confidence level matches its expected value.
- Ensure that the annotation's scenario and confidence match as intended.

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 node id and message.

Why Needed: This test prevents a bug where the structure of collection errors is not validated correctly, potentially leading to incorrect handling or reporting of these errors in the system.

Key Assertions:

- assert error.nodeid == 'test_bad.py'
- assert error.message == 'SyntaxError'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 initial state of the collector is empty.

Why Needed: This test prevents a potential regression where an empty collection is not correctly identified as such.

Key Assertions:

- The function `get_collection_errors()` should return an empty list when the input `collector` has no errors.
- No exceptions should be raised if the input `collector` does not have any errors.
- The output of `get_collection_errors()` should match the expected result (an empty list).

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 that the default value of llm_context_override is set to None when not provided.

Why Needed: This test prevents a potential bug where the default value of llm_context_override might be incorrectly set to 'None' instead of an empty string or another valid value.

Key Assertions:

- The llm_context_override field in TestCaseResult is set to None when the nodeid does not contain the word 'test'.
- The llm_context_override field in TestCaseResult is set to an empty string when the nodeid contains the word 'test'.
- The llm_context_override field in TestCaseResult is set to a valid value (e.g., 'default', 'custom') when the nodeid does not contain the word 'test' and the outcome is 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/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.

Why Needed: Prevents a regression where the default value of llm_opt_out might be set to True.

Key Assertions:

- The llm_opt_out attribute of TestCaseResult node 'test.py::test_foo' should be False.
- The llm_opt_out attribute of TestCaseResult node 'test.py::test_foo' should not be set to True.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` property of the `Config` object is set to `False` by default.

Why Needed: Without this test, the `capture` property might be incorrectly reported as `True` in certain scenarios.

Key Assertions:

- config.capture_failed_output should be False
- the `capture` property of `Config` is not set to `False` by default
- without this test, the output capture property might be misreported

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the default max chars value is not correctly set to prevent excessive output.

Key Assertions:

- assert config.capture_output_max_chars == 4000
- config.capture_output_max_chars should be equal to 4000
- The capture output should not exceed 4000 characters
- No exception should be raised when the default value is set correctly
- The `capture_output_max_chars` attribute should have a value of 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: Test verifies that xfail failures are recorded as xfailed.

Why Needed: Prevents regression where xfail failures are not properly recorded as xfailed.

Key Assertions:

- The test asserts that the collector correctly records xfail failures as xfailed.
- The test ensures that the collector logs the correct outcome for failed tests (xfailed).
- The test verifies that the collector does not incorrectly log skipped tests or other outcomes.
- The test checks that the collector's result is consistent with the expected failure outcome.
- The test verifies that the collector handles xfail failures correctly and does not silently ignore them.
- The test ensures that the collector logs the correct nodeid for the failed test.
- The test checks that the collector logs the correct when, passed, failed, skipped, duration, longrepr, and wasxfail attributes.

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 xfail passed is recorded as xpassed when an unexpected pass occurs.

Why Needed: This test prevents a regression where an unexpected pass would be incorrectly marked as failed.

Key Assertions:

- result.outcome should be 'xpassed' after the expected failure.
- report.wasxfail should be set to 'expected failure'.
- collector.results[report.nodeid] should contain the xpassed outcome.

COVERAGE

src/pytest_llm_report/collector.py	26 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-115, 124, 127, 140, 155-159, 163, 167, 171, 209-210, 212-214)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the collector is not initialized with any results, leading to incorrect assertions in subsequent tests.

Key Assertions:

- The `results` attribute of the `collector` object should be an empty dictionary.
- The `collection_errors` list should be empty.
- The `collected_count` attribute of the `collector` object should be 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 node ID.

Why Needed: This test prevents a regression where unsorted results are returned due to incorrect sorting logic in the `get_results` method.

Key Assertions:

- The list of node IDs returned by the `get_results` method is sorted in ascending order (i.e., 'a_test.py::test_a' comes before 'z_test.py::test_z').
- The node ID 'a_test.py::test_a' is present in the sorted list.
- The node ID 'z_test.py::test_z' is not present in the sorted list.
- No duplicate node IDs are present in the sorted list.
- All nodes with positive outcomes (i.e., 'passed') are included in the sorted list.
- All nodes without positive outcomes (i.e., 'failed') are excluded from the sorted list.

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: Verifies that the `handle_collection_finish` method correctly tracks collected and deselected items.

Why Needed: This test prevents a potential regression where an item is not properly tracked as collected or deselected.

Key Assertions:

- The `collected_count` attribute should be set to 3 after calling `handle_collection_finish` with the provided list of collected items.
- The `deselected_count` attribute should be set to 1 after calling `handle_collection_finish` with the provided list of deselected items.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that `capture_output` is disabled when `config.capture_failed_output=False` and `handle_report` is used for integration tests via handle_runtest_logreport.

Why Needed: This test prevents a regression where the collector captures output even if `capture_output` is disabled in the configuration, which could lead to unexpected behavior or errors.

Key Assertions:

- The `results` dictionary does not contain any captured stdout data.
- The `collector.handle_runtest_logreport(report)` method does not capture any output.
- The `result.captured_stdout` attribute is None.
- The `results` dictionary contains the expected empty dictionary.
- No captured stdout data is reported in the test results.
- The collector's output is not affected by the disabled `capture_output` configuration.

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: Verify that the `TestCollector` class captures stderr correctly.

Why Needed: This test prevents a potential bug where the `TestCollector` class does not capture stderr and instead returns an empty string.

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 does not modify the original `result` object.

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 `capture_output` method captures stdout correctly.

Why Needed: This test prevents a potential bug where the captured stdout is not included in the test result.

Key Assertions:

- The `captured_stdout` attribute of the `TestCaseResult` object should contain the expected value.
- The `report.capstdout` attribute should be set to the expected value.
- The `result.captured_stdout` attribute should match the expected value.
- The captured stdout should not be empty.
- The report should have a non-empty `capstdout` attribute.

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 `TestCollector` truncates output exceeding max chars.

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

Key Assertions:

- The captured stdout should be truncated to 10 characters.
- The captured stderr should not have any characters.
- The captured stdout should contain only digits (0-9).
- The captured stderr should be empty.
- The `TestCollector` instance is configured to capture failed output.
- The `TestCollector` instance is configured to truncate output exceeding max chars.

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 in collector._create_result method.

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

Key Assertions:

- item.callspec.id should be set to 'param1' after calling get_closest_marker('llm_opt_out')
- get_closest_marker('llm_context') should return a MagicMock object with args ['complete']
- result.llm_opt_out should be True after calling _create_result(report, item)
- result.llm_context_override should be set to 'complete' after calling get_closest_marker('llm_context')
- result.requirements should contain exactly two strings: 'REQ-1' and '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 the `extract_error` method of `TestCollector` when encountering a ReprFileLocation.

Why Needed: This test prevents potential crashes caused by using `str()` on a `ReprFileLocation` instance.

Key Assertions:

- The `_extract_error` method should return 'Crash report' as expected.
- The `report.longrepr` attribute should have been set to 'Crash report' before calling `_extract_error`.
- If `str()` is called on a `ReprFileLocation` instance, the test should raise an exception or handle it in a way that prevents crash reports.

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: The test verifies that the `_extract_error` method returns the correct string when an error occurs.

Why Needed: This test prevents a potential regression where the `longrepr` attribute is not correctly set for errors.

Key Assertions:

- The value of `report.longrepr` should be equal to 'Some error occurred'.
- The `collectors` attribute of the `collector` object should contain an instance of `TestCollector` with a `longrepr` attribute set to 'Some error occurred'.

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

Why Needed: Prevents a potential bug where the method does not handle cases without longrepr correctly.

Key Assertions:

- The `report.longrepr` attribute is set to `None` before calling `_extract_skip_reason`.
- The `_extract_skip_reason` method should return `None` in such cases.
- A test failure occurs when the expected output is `None`.
- The test case relies on the correct behavior of `_extract_skip_reason` under these conditions.
- The test ensures that the method behaves as expected without longrepr.
- No unexpected side effects occur when no longrepr is provided.
- The test does not crash or produce incorrect results due to missing longrepr.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250, 252)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test the `_extract_skip_reason` method of `TestCollector`.

Why Needed: Prevents a potential bug where the method returns an incorrect string as skip reason for longrepr.

Key Assertions:

- The method should return 'Just skipped' as expected.
- The method should not raise any exceptions when called with a valid report object.
- The method should correctly handle cases where `report.longrepr` is empty or None.
- The method should not modify the original string passed to it.
- The method should preserve the original type of `report.longrepr`.
- The method should return an empty string if no reason can be determined from `report.longrepr`.
- The method should handle cases where `report.longrepr` is a single character string.

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: Verify that `extract_skip_reason_tuple` correctly extracts skip message from a tuple longrepr.

Why Needed: Prevents potential issues where an incorrect or incomplete reason is extracted, potentially leading to misleading test results or incorrect reporting.

Key Assertions:

- The `report.longrepr` tuple contains the file name and line number as its first elements.
- The second element of the tuple is 'Skipped for reason'.
- The third element of the tuple is a string value.
- The extracted reason matches the expected one.
- No error or exception occurs during the extraction process.
- The test can be run with different input data without affecting its correctness.

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

Why Needed: This test prevents a potential bug where the collector does not handle collection reports correctly and fails to log any errors.

Key Assertions:

- The `collection_errors` list should contain exactly one element with the nodeid 'test_broken.py' and the message 'SyntaxError'.
- The value of `collector.collection_errors[0].nodeid` should be equal to 'test_broken.py'.
- The value of `collector.collection_errors[0].message` should be equal to '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 regression where the TestCollector does not handle reruns correctly, potentially leading to incorrect results or failures.

Key Assertions:

- res.rerun_count should be equal to 1 (expected)
- res.final_outcome should be 'failed' (expected)
- collector.results['t::r'] should contain the expected result

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: The test verifies that the TestCollector handles a runtest setup failure by recording an error in the report.

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

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	36 lines (ranges: 90, 93-94, 96, 99-103, 109-112, 114-115, 124, 127, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the collector does not report errors when the teardown process fails.

Key Assertions:

- res.outcome is set to 'error' after teardown failure
- res.phase is set to 'teardown'
- res.error_message contains 'Cleanup failed'

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the Gemini model parsing edge cases, specifically when no models are provided or when all models are selected.

Why Needed: This test prevents a regression where the `GeminiProvider` throws an exception when no models are specified in the configuration.

Key Assertions:

- The function `_parse_preferred_models()` should return an empty list when `None` is passed as the model argument.
- The function `_parse_preferred_models()` should return an empty list when 'All' is passed as the model 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/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: Verify that the rate limiter does not allow a request to be processed when it is over its token limit and under its request limit.

Why Needed: This test prevents a regression where the rate limiter allows requests to bypass their token or request limits, potentially leading to abuse.

Key Assertions:

- assert limiter.next_available_in(60) > 0
- assert limiter.next_available_in(10) == 0
- assert limiter.record_tokens(50) is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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` correctly returns their respective values.

Why Needed: This test prevents regression in coverage booster models where the coverage percentage is not accurately represented as a float value (e.g., due to precision issues or incorrect formatting).

Key Assertions:

- The 'coverage_percent' key in `d` should contain the expected value of 50.0.
- The 'error' key in `ann.to_dict()` should contain the expected error message 'timeout'.
- The 'duration' key in `meta.to_dict()` should contain the expected value of 1.0.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	46 lines (ranges: 71-78, 104-107, 109, 111-113, 115, 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The `CoverageMapper` class initializes correctly with a provided configuration.

Why Needed: This test prevents a potential bug where the `CoverageMapper` instance's internal state is not properly initialized.

Key Assertions:

- assert mapper.config is config
- assert mapper.warnings == []

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	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 the `get_warnings` method in `CoverageMapper` to ensure it returns a list of warnings.

Why Needed: This test prevents a potential bug where the `get_warnings` method may return an empty list if no warnings are present.

Key Assertions:

- The `warnings` variable is expected to be a list.
- The `warnings` variable is expected to contain at least one warning.
- The `warnings` variable is not empty.
- The `warnings` variable does not contain any warnings from the `CoverageMapper` instance.
- The `warnings` variable contains only warnings from the same configuration as the test.
- The `warnings` variable contains warnings that are not related to coverage.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	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 'test_map_coverage_no_coverage_file' verifies that the CoverageMapper returns an empty dictionary when no coverage file is present.

Why Needed: This test prevents a regression where the CoverageMapper does not correctly handle the case of no coverage files.

Key Assertions:

- The function `mapper.map_coverage()` should return an empty dictionary.
- The function `mapper.warnings` should have at least one warning when called.
- The `Path.exists` mock should return False for a non-existent file.
- The `glob.glob` mock should return an empty list for a non-existent glob pattern.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Verify that the `CoverageMapper` extracts node IDs for all phases when `include_phase=all`.

Why Needed: This test prevents a regression where the coverage map might not include all phases if `include_phase=all` is used.

Key Assertions:

- The method `_extract_nodeid` of the `CoverageMapper` class extracts node IDs for the specified phase.
- The method `_extract_nodeid` of the `CoverageMapper` class returns the expected node ID for each phase.
- The method `_extract_nodeid` of the `CoverageMapper` class includes all phases when `include_phase=all` is used.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	11 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

LLM error: Failed after 3 retries. Last error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	4 lines (ranges: 44-45, 216-217)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the test extracts node IDs correctly when include_phase='run' and node ID contains 'setup'.

Why Needed: This test prevents a bug where node IDs containing 'setup' are not extracted correctly due to the inclusion phase being set to 'run'.

Key Assertions:

- The function _extract_nodeid() returns None when the node ID contains 'setup'.
- The include_phase is set to 'run', which prevents the extraction of node IDs containing 'setup'.
- The node ID does not contain 'setup' for the given test.
- The node ID does not contain 'setup' and its value is extracted correctly by the CoverageMapper.
- The node ID contains 'setup' but the include_phase is set to 'run', which prevents its extraction.
- The node ID does not contain 'setup' and its value is extracted correctly by the CoverageMapper when include_phase='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/coverage_map.py	9 lines (ranges: 44-45, 216, 220, 224-225, 228-230)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the nodeid is not extracted correctly when running in the 'run' phase.

Key Assertions:

- The function `_extract_nodeid` should return the correct nodeid value.
- The nodeid value should match the expected output (`test.py::test_foo`).
- If the run phase context does not contain a `test.py::test_foo` module, 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/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 should exercise all paths in _extract_contexts to ensure full logic coverage.

Why Needed: This test prevents regression by ensuring that the mapper covers all possible paths in the codebase, including those with minimal logic.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	57 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 137-140, 144, 148, 150, 152-153, 156, 160-163, 165, 167-168, 173, 176, 178-184, 187-189, 191-194, 196, 199-200, 202, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Verify that the test extracts contexts from data with no test contexts.

Why Needed: Prevents regression in coverage reporting when there are no test contexts.

Key Assertions:

- mock_data.measured_files.return_value is an empty list
- mock_data.contexts_by_lineno.return_value is an empty dictionary
- result is 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/coverage_map.py	14 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 144-146)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test coverage for extracting node IDs in different scenarios.

Why Needed: This test prevents regression by ensuring that the `CoverageMapper` correctly identifies missing lines in `_extract_nodeid` and filters out irrelevant results.

Key Assertions:

- The function `_extract_nodeid` returns the expected node ID when the phase is 'setup'.
- The function `_extract_nodeid` returns None when the phase is 'run' or 'teardown'.
- The function `_extract_nodeid` correctly handles cases without a pipe (e.g., `test.py::test_no_phase`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	14 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231-234, 236, 239)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 'test_load_coverage_data_no_files' verifies that the test fails when no coverage files exist.

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

Key Assertions:

- The function should return None.
- There should be one warning with code 'W001'.
- The warnings list should contain only one item.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	9 lines (ranges: 44-45, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the test_load_coverage_data_read_error function handles errors reading coverage files correctly.

Why Needed: This test prevents a regression where the CoverageMapper class fails to handle corrupted coverage data files.

Key Assertions:

- The function should return None when attempting to load coverage data from a corrupt .coverage file.
- Any warnings should contain 'Failed to read coverage data' as part of their message.
- The function should not raise an exception when loading coverage data from 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/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 it correctly updates the CoverageData instances.

Why Needed: This test prevents regression in case of parallel coverage files, ensuring that the CoverageMapperMaximal class correctly handles and updates the mock CoverageData instances.

Key Assertions:

- The `update` method of the `CoverageData` instance should be called at least twice for each parallel coverage file.
- The `update` method of the `CoverageData` instance should be called only once for each non-parallel coverage file.
- The `update` method of the `CoverageData` instance should not be called for a single coverage file that is not part of any parallel process.
- The mock instances returned by `mock_data_cls.side_effect` should have different update counts for each parallel coverage file.
- The mock instances returned by `mock_data_cls.side_effect` should have the same update count for non-parallel coverage 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/coverage_map.py	15 lines (ranges: 44-45, 72-73, 83, 86, 88, 92, 94, 98, 101-104, 106)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test coverage mapping function with no data returns an empty dictionary.

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

Key Assertions:

- The _load_coverage_data method of CoverageMapper is called with None as its argument.
- The map_coverage method of CoverageMapper is called without any coverage data.
- The result of map_coverage is an empty dictionary when no coverage data is available.
- No exception is raised if the config._load_coverage_data returns None.
- The test passes even if _load_coverage_data returns a different value, such as an error.
- The test verifies that the CoverageMapper can handle missing coverage data without crashing.

COVERAGE

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

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

Key Assertions:

- mock_data.measured_files.return_value should return ['app.py']
- mock_cov.get_data.return_value should raise an exception
- mock_cov.analysis2.side_effect should be set to Exception('Analysis failed')
- entries should not contain any files with errors
- len(entries) should be equal 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/coverage_map.py	22 lines (ranges: 44-45, 243-244, 246-248, 250, 252-254, 259, 261, 263-268, 271, 299-300)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116, 123)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression by ensuring that all possible source code paths are covered, even if analysis2 is not able to provide complete information.

Key Assertions:

- The file path of the matched entry should be 'app.py'.
- The number of statements in the matched entry should be 3.
- The coverage percentage of the matched entry should be 66.67%.
- At least one statement should be missed by the matched entry.
- All covered lines should have a count greater than zero.
- All uncovered lines should have a count of zero or negative.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	32 lines (ranges: 44-45, 243-244, 246-248, 250, 252, 259-261, 273, 276-279, 281-283, 285-293, 295, 299-300)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	17 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64, 100, 103, 111-112, 116, 123)
src/pytest_llm_report/util/ranges.py	14 lines (ranges: 29, 33, 35-37, 39-40, 45-47, 50, 52, 65-66)

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

AI ASSESSMENT

Scenario: Test the `make_warning` factory function to ensure it correctly creates a Warning with the specified code and message.

Why Needed: To prevent unexpected warnings from being generated when no coverage file is found.

Key Assertions:

- The warning has the correct code (W001_NO_COVERAGE)
- The warning's message includes 'No .coverage file found'
- The warning's detail matches '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: To prevent a potential bug where the warning code values are incorrect, this test ensures they match the expected values.

Key Assertions:

- {'message': 'Assertion failed: WarningCode.W001_NO_COVERAGE.value == "W001"', 'description': 'Expected value of WarningCode.W001_NO_COVERAGE to be "W001"'}
- {'message': 'Assertion failed: WarningCode.W101_LLM_ENABLED.value == "W101"', 'description': 'Expected value of WarningCode.W101_LLM_ENABLED to be "W101"'}
- {'message': 'Assertion failed: WarningCode.W201_OUTPUT_PATH_INVALID.value == "W201"', 'description': 'Expected value of WarningCode.W201_OUTPUT_PATH_INVALID to be "W201"'}
- {'message': 'Assertion failed: WarningCode.W301_INVALID_CONFIG.value == "W301"', 'description': 'Expected value of WarningCode.W301_INVALID_CONFIG to be "W301"'}
- {'message': 'Assertion failed: WarningCode.W401_AGGREGATE_DIR_MISSING.value == "W401"', 'description': 'Expected value of WarningCode.W401_AGGREGATE_DIR_MISSING to be "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 Warning.to_dict() method to ensure it returns a dictionary with the correct keys.

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

Key Assertions:

- The 'code' key should be present in the dictionary and have the value 'W001'.
- The 'message' key should be present in the dictionary and have the value 'No coverage'.
- The 'detail' key should be present in the dictionary and have the value 'some/path'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	6 lines (ranges: 70-72, 74-76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test the `make_warning` function with known code.

Why Needed: Prevents a potential warning that may not be caught by other checks.

Key Assertions:

- The `make_warning` function should create a warning with the correct code and message.
- The warning message should match the expected value `WARNING_MESSAGES[WarningCode.W101_LLM_ENABLED]`.
- The detail field of the warning object should be None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test MakeWarning for unknown code when enum is not allowed

Why Needed: Prevents a potential bug where the test fails due to an unknown warning message being used.

Key Assertions:

- The function `make_warning` should be able to handle missing messages from the `WARNING_MESSAGES` dictionary.
- The assertion `w.message == 'Unknown warning.'` should pass even if the actual value is different.
- The comment in the code suggests that an unknown message should be used, but this test verifies it doesn't happen.
- If the enum was extended or a key was missing from the dictionary, this test would fail
- The `try-except-finally` block ensures that the old warning message is restored after the test finishes

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test 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 generated when a configuration issue is detected.

Key Assertions:

- w.code == WarningCode.W301_INVALID_CONFIG
- w.detail == 'Bad value'
- assert w.detail in ['Bad value', 'Invalid configuration']
- assert isinstance(w, Warning)
- assert len(w.detail) > 0
- assert w.detail != ''
- assert w.detail.startswith('Bad value') or w.detail.startswith('Invalid configuration')

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Testing whether WarningCode enum values are correctly identified as strings.

Why Needed: This test prevents a potential bug where the WarningCode enum is not properly validated against string types.

Key Assertions:

- assert isinstance(code.value, str) checks if the value of each code in the WarningCode enum is indeed a string.
- assert code.value.startswith('W') checks if the value starts with 'W' as expected for WarningCode enum 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 warning_to_dict method of Warning class returns a dictionary with required keys.

Why Needed: This test prevents a potential bug where the warning is not properly serialized to a dictionary without including the detail message.

Key Assertions:

- The 'code' key should be set to 'W001'.
- The 'message' key should be set to 'No coverage'.
- The 'detail' key should be an empty string or None. If it's not, this test will fail.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	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 objects can be serialized to a dictionary with detail.

Why Needed: Prevents a warning from being silently ignored when converting Warning objects to dictionaries.

Key Assertions:

- The 'code' key should contain the correct value.
- The 'message' key should contain the correct value.
- The 'detail' key should contain the correct value and be set to 'Check setup'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/errors.py`	`6 lines (ranges: 70-72, 74-76)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

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

AI ASSESSMENT

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

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

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 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 return True for a file named `foo/bar.py`.
- The function should correctly identify `.py` files.
- The function should not incorrectly identify other types of files (e.g. `foo.txt`, `bar.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/util/fs.py	1 lines (ranges: 79)

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

AI ASSESSMENT

Scenario: Test makes absolute path relative to test directory.

Why Needed: Prevents regression where `make_relative` function is called with a non-existent parent directory.

Key Assertions:

- The file system should create the parent directory if it does not exist.
- The file system should be able to touch the created file.
- The `make_relative` function should return the expected relative path.
- The `parent` attribute of the resulting file path should be set correctly.
- The `exist_ok` parameter should have no effect on the creation of the parent directory.
- The `touch` method should not raise an exception if the file already exists.
- The `mkdir` method should create the parent directory and all its parents if they do not exist.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/fs.py`	`11 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64)`

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where an incorrect normalization may occur when no base is provided.

Key Assertions:

- The output of `make_relative('foo/bar')` should be 'foo/bar'.
- The input path should not have any leading slashes before the relative path.
- The function should handle cases where the input path starts with a slash.
- No error or exception should be raised when no base is provided.
- The output of `make_relative('/path/to/relative/path')` should also be 'foo/bar'.
- The function should not modify the original path in any way.
- The relative path should be appended to the end of the input path without changing its position.

COVERAGE

src/pytest_llm_report/collector.py	14 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: Tests that the `normalize_path` function handles already-normalized paths correctly.

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

Key Assertions:

- The input path should be normalized before being passed to the `normalize_path` function.
- The normalized path should be returned by the `normalize_path` function.
- Any modifications made to the input path after normalization should not affect its original state.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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 `normalize_path` function with forward slashes in file paths.

Why Needed: Prevents a potential bug where the function does not correctly handle forward slashes in Windows file paths.

Key Assertions:

- The function should convert backslashes (`\`) to forward slashes (`/`).
- The resulting path should be in the format `foo/bar` rather than `foo␀ar`.
- The function should handle both single and double backslashes correctly.
- The function should not throw any errors for invalid or unsupported backslash sequences.
- The function should preserve the original directory structure of the input path.
- The function should be able to handle paths with multiple consecutive forward slashes.
- The function should not convert consecutive forward slashes into a single forward slash.
- The function should correctly handle Windows-style file separators in the current working directory.

COVERAGE

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

Why Needed: Prevents the test from failing if a path ends with a slash.

Key Assertions:

- The function `normalize_path` removes any trailing slashes from the input path.
- The function `normalize_path` returns the normalized path as a string.
- The function `normalize_path` handles paths with leading or trailing slashes correctly.
- If the input path does not end with a slash, the function should return the original path.
- The function should handle cases where the input is an empty string or a single slash.
- The function should handle cases where the input contains multiple consecutive slashes.
- The function should preserve the directory hierarchy of the input path.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	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 function incorrectly includes all files in the excluded patterns, leading to false positives or incorrect skipping of certain paths.

Key Assertions:

- assert should_skip_path('tests/conftest.py', exclude_patterns=['test*']) is True
- assert should_skip_path('src/module.py', exclude_patterns=['test*']) 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	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: This test prevents a regression where the 'should_skip_path' function incorrectly skips normal paths.

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 regression where the function incorrectly skips non-existent `.git` directories.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the function might incorrectly skip certain directories or files, potentially causing unexpected behavior in other parts of the codebase.

Key Assertions:

- The function should return `True` for the given path.
- The function should raise an error with a specific message if the path is not a valid __pycache__ directory.
- The function should skip any files that are not in the correct format (i.e., `.pyc` or `.pyo`).
- The function should handle cases where the `__pycache__` directory is empty.
- The function should correctly handle cases where the path does not exist.
- The function should raise an error if the path is a non-ASCII file (e.g., `.pyc` or `.pyo`).
- The function should skip any directories that are already skipped by `should_skip_path`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function `should_skip_path` is called with the string 'venv/lib/python/site.py' as an argument.
- The function `should_skip_path` is called with the string '.venv/lib/python/site.py' as an argument.
- The function `should_skip_path` returns True for both arguments.
- The function `should_skip_path` does not include 'venv' in its list of paths to skip.
- The function `should_skip_path` correctly handles the case where the path is a subdirectory of another directory.
- The function `should_skip_path` raises an error if it encounters an invalid argument.
- The function `should_skip_path` does not include 'site.py' in its list of paths to skip for venv directories.

COVERAGE

src/pytest_llm_report/collector.py	14 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 removes request and token usage logs after a certain time in the past.

Why Needed: This test prevents regression where the rate limiter logs requests and tokens used before a certain time, potentially causing issues with logging or analytics.

Key Assertions:

- The length of _request_times should be 0 after pruning.
- _token_usage should have zero values after pruning.
- limiter._request_times should contain only one element (the timestamp) after pruning.
- limiter._token_usage should contain only one tuple (the timestamp and usage count) after pruning.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	11 lines (ranges: 39-42, 81-85, 87-88)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents requests from exceeding a certain threshold (1-3 sentences)

Why Needed: This test prevents a potential bug where the rate limiter allows too many requests within a minute, leading to performance issues or unexpected behavior.

Key Assertions:

- The `next_available_in` method should return a value greater than 0
- The `next_available_in` method should return a value less than or equal to 60.0 (1-2 seconds)
- The rate limiter should be unavailable after the first request is 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	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 immediately available.

Why Needed: This test prevents a bug where the rate limiter does not prevent token updates from being recorded even after tokens have been consumed.

Key Assertions:

- The next_available_in method returns a non-zero value (20) indicating that there is still available capacity in the rate limiter.
- The _token_usage list indicates that only two tokens were actually used.
- The record_tokens method updates the state of the rate limiter correctly, preventing further token updates from being 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	33 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 81-82, 84, 87-88, 92-94, 100-101, 103, 105, 107-108, 110-114)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `wait_for_slot` method should be called with the specified argument (1).
- The `time.sleep` mock object should have been called with the expected argument (1).
- The `time.sleep` mock object should not have been called before the first request is recorded.
- The `time.sleep` mock object should not have been called after the last request has been recorded.
- The `wait_for_slot` method should return immediately if no requests are pending.
- The `wait_for_slot` method should raise an exception if it encounters a problem while waiting for a slot.
- The `requests_per_minute` argument is validated correctly (1).
- The `GeminiRateLimiter` instance has the correct attributes 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/llm/gemini.py	31 lines (ranges: 39-42, 45-46, 48, 52-54, 58-59, 61-63, 73, 76-78, 81-82, 84, 87-88, 92-93, 95, 97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the `record_tokens` method of `_GeminiRateLimiter` returns early when no tokens are available.

Why Needed: Prevents a potential regression where the limiter does not record tokens for zero tokens, potentially leading to incorrect usage statistics.

Key Assertions:

- The length of `_token_usage` is 0 after calling `record_tokens(0)`.
- The value of `_token_usage` should be an empty list.
- _token_usage is a list containing the expected number of tokens.
- The limiter's usage statistics are correct for zero tokens.
- _token_usage is not equal to [0].
- The limiter's usage statistics are correct when no tokens are available.
- The length of `_token_usage` remains 0 after calling `record_tokens(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/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 RateLimitExceeded exception when exceeding the daily limit.

Why Needed: This test prevents a potential rate limiting error where an excessive number of requests are made in a short period, potentially causing the application to become unresponsive or experience performance issues.

Key Assertions:

- The limiter raises a RateLimitExceeded exception with the message 'requests_per_day':
- The exception is raised within 10 seconds after the request is made.
- The error message contains the string 'requests_per_day'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	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 wait time fallback works correctly when filling up the rate limiter.

Why Needed: The current implementation may cause a `TimeOut` exception if the TPM is fully filled before the request can be processed due to excessive token usage.

Key Assertions:

- The value of `wait` should be greater than 0 after filling up the rate limiter.
- The number of tokens used plus the requested tokens should exceed the rate limit.
- Token usage is not empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	24 lines (ranges: 39-42, 66, 68-70, 81-82, 84, 87-88, 100-101, 103, 105, 107-108, 110-114)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_gemini_coverage_v2.py::test_gemini_provider_rpm_cooldown 546ms 6

AI ASSESSMENT

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

Why Needed: To prevent the test from failing due to a missing cooldown after an initial failed call to RPM rate limit.

Key Assertions:

- The 'models/gemini-pro' model should be present in the provider's cooldowns.
- The cooldown for 'models/gemini-pro' should be greater than 1000.0 seconds (1 minute).
- The provider should have a valid 'cooldowns' dictionary with 'models/gemini-pro' as a key.
- The provider should not fail immediately after an initial failed call to RPM rate limit, but instead retry and succeed.

COVERAGE

src/pytest_llm_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 case of rate limit retries, ensuring that the provider can handle such scenarios without failing.

Key Assertions:

- The annotation is set to "Recovered Scenario" even after a 429 response from the API.
- The second call to `_annotate_internal` with the same `test_result` object succeeds with a 200 status code.
- The number of calls to `mock_post` is exactly 2, as expected for a retry scenario.
- The annotation's `scenario` attribute matches the expected value "Recovered Scenario".
- No error is reported in the annotation's `error` attribute.
- A small sleep mock has been used to speed up the test, but still allows the test to complete successfully.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	181 lines (ranges: 32-34, 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-215, 217-222, 225-227, 233-234, 238-240, 242-243, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330-333, 335-341, 343, 346, 348-350, 352-355, 360-363, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that _annotate_success returns the correct scenario and no error when successful.

Why Needed: This test prevents a potential regression where the provider might return an incorrect or missing scenario for success scenarios.

Key Assertions:

- The annotation returned by _annotate_internal has the expected scenario.
- The annotation does not have any errors.
- The annotation is of type LlmAnnotation and its scenario matches the provided scenario.
- The annotation is not None and has no error.
- _parse_response returns a Mock object with the correct scenario and no error.
- The _parse_response method calls _annotate_internal correctly.
- The _parse_response method does not raise an exception when called successfully.

COVERAGE

src/pytest_llm_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: Tests the availability of the GeminiProvider when it's not configured to be available.

Why Needed: This test prevents a potential bug where the provider tries to access an unavailable resource.

Key Assertions:

- The provider should return False indicating that it's not available.
- The provider should have its `_check_availability` method set to `False` when no environment variable is present.
- The provider should have a valid configuration with the correct provider name.
- The provider should be able to access an unavailable resource using the provided API token.
- The provider's `_check_availability` method should return True after setting the environment variable back to its original 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/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 excessive requests within a certain time frame.

Why Needed: This test prevents a potential bug where an attacker could flood the API with requests and exceed the daily limit.

Key Assertions:

- The next_available_in method returns None when there are no available slots in the rate limiter.
- The limiter does not allow more than one request per second.
- The limiter does not allow more than two requests within a minute.
- The limiter does not allow more than five requests within an hour.
- The next_available_in method returns None when there are no available slots in the rate limiter after 100 requests.
- The limiter allows one request per day.
- The limiter does not allow any requests during a 24-hour period.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	18 lines (ranges: 39-42, 45-46, 48-50, 73, 76-78, 81-82, 84, 87-88)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the rate limiter does not allow more than 2 requests per minute for a single user.

Why Needed: This test prevents a potential bug where the rate limiter allows too many requests from a single user, potentially leading to performance issues or even DDoS attacks.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	27 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: This test prevents a potential bug where two different configuration providers could yield the same hash, potentially leading to unexpected behavior or errors in downstream systems.

Key Assertions:

- Two Config objects with different 'provider' attributes should have different 'hash' values.
- The 'compute_config_hash()' function should return a different value for config1 and config2.
- The 'compute_config_hash()' function should raise an AssertionError if both config1 and config2 are identical.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 96-101, 103-104)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential issue where the hash might be too long, potentially causing issues with storage or comparison operations.

Key Assertions:

- The length of the computed hash should not exceed 15 characters.
- The first 15 characters of the hash should be '16' (or close to it).
- No leading zeros are present in the hash.
- The hash does not contain any non-ASCII characters.
- The hash is a valid hexadecimal 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	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 96-101, 103-104)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression in cases where the file contents change but the file path remains the same, causing unexpected differences in the computed hashes.

Key Assertions:

- The computed SHA-256 hash of the file should be equal to its content hash.
- The content hash of the file should match the computed SHA-256 hash.
- The computed SHA-256 hash of the file should not change even if the input is modified (e.g., by renaming or deleting the 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/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 that the computed SHA-256 hash of a file matches its expected value.

Why Needed: Prevents potential issues with file integrity and data consistency.

Key Assertions:

- The length of the computed hash should be exactly 64 bytes.
- The computed hash should match the expected hash of the provided file contents.
- Any changes to the file contents should not affect the computed hash.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 the case where two different keys are used to compute a hash.

Why Needed: This test prevents potential issues with key collision and ensures that different keys produce unique signatures.

Key Assertions:

- The computed signature for `sig1` is different from the computed signature for `sig2`.
- The computed signature for `sig1` does not match any known value of `sig2`.
- The computed signature for `sig2` does not match any known value of `sig1`.

COVERAGE

src/pytest_llm_report/collector.py	14 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 the HMAC length is not as expected due to incorrect key size or mismatched padding.

Key Assertions:

- The length of the generated HMAC signature should be 64 bytes.
- The length of the generated HMAC signature should match the expected value (64 bytes).
- The length of the generated HMAC signature should not exceed the expected value (64 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/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 input produces the same output.

Why Needed: Prevents a bug where different inputs produce different hashes, potentially leading to data corruption or inconsistencies.

Key Assertions:

- The two computed hashes are equal.
- The two hashed strings have the same hexadecimal representation.
- The two hashed strings have the same SHA-256 digest (hash value).
- The input string 'test' is the same across both computations.
- The output of `compute_sha256(b'test')` and `compute_sha256(b'test')` are the same.
- The hash of a string with no modifications remains unchanged.
- The hash of an empty byte string is an empty byte string.
- The hash of a string containing only whitespace characters remains unchanged.

COVERAGE

src/pytest_llm_report/collector.py	14 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.

Why Needed: Prevents a potential issue where the hash is not 64 characters long, potentially causing incorrect data integrity checks.

Key Assertions:

- The length of the computed hash should be exactly 64 hexadecimal characters.
- The hash value should not be shorter than 64 characters.
- The hash value should not exceed 64 characters.
- The hash value should have all hexadecimal digits (0-9, a-f, A-F).
- No leading zeros are allowed in the hash value.
- No trailing zeros are allowed in the hash value.
- All characters in the hash value must be either hexadecimal digits or spaces.
- The hash value is not empty.

COVERAGE

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

Key Assertions:

- The 'pytest' package should be present in the `get_dependency_snapshot()` function's output.
- The 'pytest' package should be listed as one of the dependencies in the snapshot.
- The presence of 'pytest' in the snapshot indicates that it is correctly imported and available for use.
- Without 'pytest', the dependency snapshot would not accurately reflect the project's dependencies.
- Including 'pytest' ensures that tests are able to import and use the package without 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/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: Verifies that the `get_dependency_snapshot()` function returns a dictionary.

Why Needed: Prevents a potential bug where the function might return an incorrect data type or format.

Key Assertions:

- snapshot is an instance of dict
- snapshot has no attributes other than those expected by the test (e.g., 'dependencies', 'versions')
- the keys in snapshot are present and match the expected keys (e.g., 'dependencies' and 'versions')
- all values in snapshot are present and match the expected values (e.g., list of dependencies and dictionary of versions)
- snapshot does not contain any unexpected or null 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/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 loaded key is not correct due to incorrect encoding or formatting of the file.

Key Assertions:

- The `load_hmac_key` function should be able to read the key from the file and return it as bytes.
- The `load_hmac_key` function should correctly handle any errors that may occur while reading the file.
- The `load_hmac_key` function should not throw an exception if the file is empty or does not contain a valid key.
- The `load_hmac_key` function should preserve the original encoding of the file (e.g. UTF-8).
- The `load_hmac_key` function should handle any invalid characters in the file correctly (e.g. non-printable 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	5 lines (ranges: 73, 76-77, 80-81)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the function may return an incorrect result or raise an exception if a missing key file is encountered.

Key Assertions:

- The function `load_hmac_key(config)` should be able to successfully load the HMAC key from the provided configuration, even if the key file does not exist.
- The test should fail when a non-existent key file is passed as an argument to the `Config` constructor.
- The `key` variable should be set to `None` after calling `load_hmac_key(config)` in this scenario.
- The function should raise a `FileNotFoundError` if the key file does not exist, rather than returning None or raising an exception.
- The test should only fail when the key file is nonexistent, and not when it's missing a specific key (e.g. 'secret.key')
- The configuration object passed to `load_hmac_key(config)` should still be valid even if the key file 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	4 lines (ranges: 73, 76-78)

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

AI ASSESSMENT

Scenario: Verify that the test returns None when no key file is specified.

Why Needed: Prevents a bug where the function does not return an error message or a specific value when no key file is provided.

Key Assertions:

- The function should raise a ValueError with a descriptive message if no key file is specified.
- The function should return None as expected for invalid input.
- The function should handle the case where the load_hmac_key method raises an exception and propagate it correctly.
- The function should not attempt to use the loaded key even though no key file was provided.
- The function's behavior should be consistent across different test runs with varying 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/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: Verifies 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
- config.aggregate_policy == 'latest'
- config.aggregate_include_history is False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the default value of `capture_failed_output` is set to False in the `get_default_config()` function.

Why Needed: This test prevents a bug where the default value of `capture_failed_output` might be incorrectly set to True, potentially leading to unexpected behavior or errors.

Key Assertions:

- The `capture_failed_output` attribute of the returned configuration object is `False`.
- The `capture_failed_output` attribute of the returned configuration object does not have a default value of `True`.
- The `capture_failed_output` attribute of the returned configuration object is set to `None` instead of `False` when it should be `False`.
- The `capture_failed_output` attribute of the returned configuration object has a different type than expected (bool) when it should be a boolean value.
- The `capture_failed_output` attribute of the returned configuration object does not match its current value when it should match its default value.
- The `capture_failed_output` attribute is not set to `False` in the test case's expected output.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/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: Verifies that the context mode of the default LLM is set to 'minimal' when no other context mode is specified.

Why Needed: The test prevents a potential bug where the context mode defaults to 'minimal' without any explicit specification.

Key Assertions:

- config.llm_context_mode == 'minimal'
- assert isinstance(config.llm_context_mode, str)
- assert config.llm_context_mode in ['minimal', 'none']
- assert get_default_config().llm_context_mode != 'minimal'
- assert get_default_config().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: Verify that LLM is not enabled by default in the configuration.

Why Needed: Prevents regression where LLM is enabled by default, ensuring consistent behavior across tests.

Key Assertions:

- The `is_llm_enabled()` method returns False for the default configuration.
- The `llm_enabled` attribute of the default configuration does not contain 'True'.
- The default configuration does not include any LLM-related settings.
- The `LLM` key in the configuration dictionary is missing or empty.
- The `is_llm_enabled()` method returns False when called on a non-default configuration.
- The `llm_enabled` attribute of a non-default configuration contains 'False'.
- A default configuration does not include any LLM-related settings that could prevent LLM from being 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/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 provides a default configuration with `omit_tests_from_coverage` set to `True`.

Why Needed: This test prevents regression where the coverage analysis does not include tests by default.

Key Assertions:

- The `config.omit_tests_from_coverage` attribute is set to `True`.
- The `get_default_config()` function returns a configuration with `omit_tests_from_coverage` set to `True`.
- The `assert` statement checks that the `config` object has an attribute named `omit_tests_from_coverage` and its value is `True`.
- The `config.omit_tests_from_coverage` property of the `get_default_config()` function returns a boolean value.
- The `config.omit_tests_from_coverage` attribute is set to `True` in the `TestConfigDefaults` class's default configuration.
- The `TestConfigDefaults` class provides a default configuration with `omit_tests_from_coverage` set to `True`.
- The `get_default_config()` function returns a configuration that meets 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/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: Verify that the provider defaults to 'none' when None is provided.

Why Needed: The test prevents a potential bug where the provider is set to 'none' without any privacy settings being applied.

Key Assertions:

- config.provider == 'none'
- assert config.provider != 'default'
- assert config.provider != 'public'
- assert config.provider != 'private'
- assert config.provider != 'shared'
- assert config.provider != 'custom'
- assert config.provider == None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that secret files are excluded by default in the test configuration.

Why Needed: This test prevents a regression where the default test configuration does not exclude secret files from being processed.

Key Assertions:

- The 'llm_context_exclude_globs' key in the config dictionary contains any strings containing 'secret'.
- The 'llm_context_exclude_globs' key in the config dictionary contains any strings containing '.env'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verifies that the deterministic output of the test is reported correctly.

Why Needed: To prevent a regression where the report may not be sorted by nodeid due to changes in the test data.

Key Assertions:

- The `nodeids` list should contain only unique values.
- The `nodeids` list should be sorted in ascending order.
- Any duplicate nodeids should be removed from the list.
- The sorted list of nodeids should match the original list.
- No duplicates should be present in the sorted list of nodeids.
- All nodes with a unique value should have a corresponding nodeid in the sorted list.
- Nodes without a unique value 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	78 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	117 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where an empty test suite would produce incorrect or incomplete reports.

Key Assertions:

- The total count of tests in the report is zero.
- The summary section of the report contains no data.
- All test cases are successfully executed without errors.
- No error messages are displayed when running the test.
- The report includes a 'Summary' section with a 'Total' key set to 0.
- The report does not contain any failed or skipped tests.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	67 lines (ranges: 229-231, 233, 235, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	118 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

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

AI ASSESSMENT

Scenario: The full pipeline generates an HTML report.

Why Needed: This test prevents a regression where the HTML report is not generated correctly.

Key Assertions:

- The file 'report.html' exists at the expected location.
- The content of the 'report.html' file contains the string '
- The string 'test_pass' is present in the content of the 'report.html' file, indicating that a test 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/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` 56ms 7

AI ASSESSMENT

Scenario: The test verifies that a full pipeline generates a valid JSON report.

Why Needed: This test prevents regression in the generation of JSON reports for pipelines with multiple tests.

Key Assertions:

- The 'schema_version' key in the generated JSON report matches the expected value.
- The total count of passed, failed, and skipped tests is accurate according to the test results.
- At least one test was marked as passed, at least one was marked as failed, and at least one was marked as skipped.

COVERAGE

src/pytest_llm_report/_git_info.py	2 lines (ranges: 2-3)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	78 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	133 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-322, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 376, 378-379, 382,

	385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

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

AI ASSESSMENT

Scenario: Test that the `ReportRoot` object has all required fields when created.

Why Needed: This test prevents a potential bug where a missing field would cause an error during report processing.

Key Assertions:

- The `schema_version` field is present in the `data` dictionary.
- The `run_meta` field is present in the `data` dictionary.
- The `summary` field is present in the `data` dictionary.
- The `tests` field is 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	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 `RunMeta` has an 'aggregation_fields' key.

Why Needed: Prevents regression where the schema does not have aggregation fields.

Key Assertions:

- is_aggregated is present in the data.
- run_count is present in the data.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`29 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

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

AI ASSESSMENT

Scenario: Test `test_run_meta_has_status_fields` verifies that the RunMeta object has status fields.

Why Needed: This test prevents a potential bug where the RunMeta object does not have all required status fields.

Key Assertions:

- status fields should be present in the data
- exit_code field should be present in the data
- interrupted field should be present in the data
- collect_only field should be present in the data
- collected_count field should be present in the data
- 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 semver-like format.

Why Needed: This test prevents regression where the schema version is not defined or does not match a semver-like format, potentially causing issues with integration gate functionality.

Key Assertions:

- SCHEMA_VERSION must be an instance of `int` or `float` and should contain at least one dot (.)
- The value of `SCHEMA_VERSION` should be in the range of 1.0 to 5.0

COVERAGE

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

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

AI ASSESSMENT

Scenario: The `test_test_case_has_required_fields` test verifies that the `TestCaseResult` object has all required fields.

Why Needed: This test prevents a potential bug where a `TestCaseResult` object is created without providing all necessary information, potentially causing issues with downstream processing or validation.

Key Assertions:

- The 'nodeid' field should be present in the `data` dictionary.
- The 'outcome' field should also 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 provider is not correctly identified as 'GeminiProvider' even if the correct model is used.

Key Assertions:

- The `get_provider` function should return an instance of GeminiProvider when the 'provider' parameter is set to 'gemini'.
- The `__class__.__name__` attribute of the returned provider instance should match 'GeminiProvider'.

COVERAGE

src/pytest_llm_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 'litellm' provider is specified in the configuration.

Why Needed: This test prevents a potential bug where the `get_provider` function does not return an instance of LiteLLMProvider when a 'litellm' provider is specified, potentially causing unexpected behavior or errors.

Key Assertions:

- The `provider` attribute of the returned `LiteLLMProvider` instance should be equal to 'liteLLM'.
- The `__class__.__name__` attribute of the returned `LiteLLMProvider` instance should be equal to 'LiteLLMProvider'.
- The `get_provider` function is called with a valid configuration object that specifies a 'litellm' provider.
- The `get_provider` function returns an instance of LiteLLMProvider when a 'litellm' provider is specified in the configuration.
- The `provider` attribute of the returned `LiteLLMProvider` instance is not empty or None.
- The `__class__.__name__` attribute of the returned `LiteLLMProvider` instance is not equal to 'LiteLLMProvider'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	9 lines (ranges: 52-53, 245, 247, 249, 252, 257-258, 260)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the `GetProvider` function returns a `NoopProvider` when provided with 'none' as the configuration.

Why Needed: This test prevents a potential regression where the `GetProvider` function does not return a `NoopProvider` when given an invalid or missing provider name.

Key Assertions:

- The `provider` attribute of the returned `NoopProvider` instance is set to 'none'.
- The type of the `provider` attribute is indeed `NoopProvider`.
- The `get_provider` function returns a `NoopProvider` instance with the specified configuration.
- The `GetProvider` function correctly handles cases where an invalid or missing provider name is provided.
- The test does not fail when given an invalid or missing provider name, such as 'invalid' or an empty string.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 `provider='ollama'`, the function `get_provider(config)` returns an instance of OllamaProvider.

Why Needed: This test prevents a potential bug where the provider is not correctly identified as 'ollama'.

Key Assertions:

- assert provider.__class__.__name__ == 'OllamaProvider'
- assert isinstance(provider, get_provider)
- assert provider.model == 'llama3.2'
- assert provider.provider == 'ollama'
- assert provider.config is not None
- assert provider.config.provider == 'ollama'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 245, 247, 249, 252-253, 255)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that an unknown provider raises a ValueError.

Why Needed: To prevent the test from passing when an unknown provider is used.

Key Assertions:

- The function `get_provider` should be called with a valid provider.
- The function `get_provider` should raise a ValueError for an unknown provider.
- The error message should contain 'unknown' as a keyword.
- The error message should not contain any other keywords.
- The error message should be case-insensitive and contain 'unknown'.
- The exception raised by the function should have the string 'unknown' in its message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 245, 247, 249, 252, 257, 262, 267)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that NoopProvider implements LlmProvider interface.

Why Needed: Prevents regression in the implementation of LlmProvider contract.

Key Assertions:

- provider should have required methods
- provider should be able to annotate data
- provider should be available for use
- provider should have a model name attribute
- provider should have a 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 even when there is no need for one.

Key Assertions:

- The annotation should be of type LlmAnnotation
- The scenario attribute should be an empty string
- The why_needed attribute should be an empty string
- There should be an empty list of key assertions

COVERAGE

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

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

AI ASSESSMENT

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/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 always available.

Why Needed: Prevents a potential bug where the provider might not be available due to external dependencies or configuration issues.

Key Assertions:

- The `is_available()` method should return True for all instances of the NoopProvider.
- The `is_available()` method should raise an AssertionError if it returns False.
- The `provider` attribute should have a property `available` that is set to True.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Tests annotation functionality with summary emission.

Why Needed: This test prevents regression where the annotation process does not emit a summary.

Key Assertions:

- The function `test_annotate_tests_emits_summary` is called with a valid configuration and no errors.
- The `provider` attribute of the `TestCaseResult` object is set to a valid instance of `FakeProvider`.
- The `llm.annotator.get_provider` method is called with a valid configuration, resulting in a valid provider instance.
- The `annotate_tests` function is called with a list of test objects and a configured provider instance, without any errors or exceptions.
- The captured output from the `capyss.readouterr()` call contains the expected string 'Annotated 1 test(s) via litellm'.
- No other assertions are performed in this test function.

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 LLM annotator reports progress via a callback.

Why Needed: This test prevents regression where the LLM annotator does not report progress or is not properly configured to do so.

Key Assertions:

- The test verifies that the LLM annotation starts with the expected message.
- The test verifies that the correct test case is reported in the progress message.
- The test verifies that the provider used for LLM annotations is correctly set up and configured.
- The test verifies that the messages appended to the config are properly formatted and contain the expected information.
- The test verifies that the LLM annotation starts with a specific message indicating the number of tests being annotated.
- The test verifies that the correct test case is reported in the progress message.

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: This test prevents regression where LLM annotations are not respecting the opt-out and limit settings.

Key Assertions:

- The 'llm_opt_out' attribute of each TestCaseResult should be set to False.
- The 'llm_max_tests' attribute of the Config object should be set to 1.
- The provider function should call FakeProvider with a valid LlmAnnotation instance.
- The first TestCaseResult's llm_annotation should not be None.
- The second TestCaseResult's llm_annotation should be None.
- The third TestCaseResult's llm_annotation should also be None.

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 and fails to report the correct tests.

Key Assertions:

- provider.calls should contain all test nodes.
- sleep_calls should be equal to the number of times the monotonic function was called.
- the time between each sleep call is exactly 2.0 seconds.
- all calls made by the provider should have been recorded in the tests.
- the total number of calls made by the provider should match the expected list of test nodes.
- the sleep calls should be equal to the expected number of times the monotonic function was called.
- the time between each sleep call should be exactly 2.0 seconds.

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 skips unavailable providers with a clear message.

Why Needed: This test prevents regression where annotation fails due to unavailable provider.

Key Assertions:

- The function `annotate_tests` should skip the annotation process when an unavailable provider is detected.
- The message 'is not available' should be logged in the captured output.
- The annotation process should fail without producing any output for unavailable 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/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: The test verifies that annotations are cached between runs.

Why Needed: This test prevents a regression where the annotator might not be able to cache annotations, potentially leading to inconsistencies in the results.

Key Assertions:

- provider.calls == ['tests/test_sample.py::test_case']
- test.llm_annotation is not None
- test.llm_annotation.scenario == 'cached'

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: This test verifies that the `test_required_fields` function checks for the presence of 'scenario' and 'why_needed' keys.

Why Needed: The schema requires these fields to be present, which prevents a potential bug where the test fails if they are missing.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: This test verifies that the `AnnotationSchema.from_dict` method correctly parses a dictionary into an AnnotationSchema instance.

Why Needed: The `AnnotationSchema.from_dict` method prevents authentication bypass by ensuring that password and username checks are performed before returning the schema.

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 is not properly initialized with an empty dictionary.

Key Assertions:

- schema.scenario = "" (empty string)
- schema.why_needed = "" (empty string) (to indicate that no specific reason is needed for this scenario)
- schema.input_type = "dict" (AnnotationSchema.from_dict is used to create a new schema from an empty dictionary)
- schema.output_type = "str" (AnnotationSchema output type should be str)
- schema.validate_input = False (AnnotationSchema validation input is set to False when it's empty)
- schema.validate_output = True (AnnotationSchema validation output is set to True for non-empty 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/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where an incorrect or missing scenario attribute is set in the AnnotationSchema.from_dict method for partial inputs.

Key Assertions:

- The schema attribute should be set to 'Partial only'.
- The why_needed attribute should be set to an empty string.
- The annotation should have a valid scenario attribute when passed as 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/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 bug where the schema does not have all necessary fields, potentially leading to incorrect validation or errors.

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

Why Needed: This test prevents regression by ensuring that the AnnotationSchema instance can be properly converted into a dictionary.

Key Assertions:

- assertion 1: The 'scenario' key in the data dictionary matches the expected value.
- assertion 2: The 'why_needed' key in the data dictionary matches the expected value.
- assertion 3: The 'key_assertions' list in the data dictionary contains all the 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: The factory function should return a NoopProvider instance when the specified provider is 'none'.

Why Needed: This test prevents a potential bug where the factory returns an incorrect provider.

Key Assertions:

- the function config.provider returns 'none'
- the function get_provider(config) returns a NoopProvider instance
- the assert isinstance(provider, NoopProvider) passes

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 245, 247, 249-250)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `NoopProvider` class might not implement the required interface for `LlmProvider`.

Key Assertions:

- The `provider` variable should be assigned an instance of `LlmProvider`.
- The `provider` variable should be assigned an instance of `LlmProvider` with the correct attributes.
- The `provider` variable should not throw any exceptions when instantiated.
- The `provider` variable should have a valid `__class__` attribute.
- The `provider` variable should have a valid `__name__` attribute.
- The `provider` variable should be an instance of the correct class.
- The `provider` variable should not inherit from another class.
- The `provider` variable should not have any additional attributes.

COVERAGE

src/pytest_llm_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: NoopProvider returns empty annotation when no function is annotated with @noop.

Why Needed: This test prevents a regression where the NoopProvider would return an annotation for a function that does not contain any @noop decorators.

Key Assertions:

- The annotation returned by the NoopProvider should be empty.
- The scenario of this test is empty.
- No key assertions were performed in this test.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that annotate function returns the correct annotation object with required attributes.

Why Needed: This test prevents a regression where the annotate function does not return an LlmAnnotation-like object with the expected attributes.

Key Assertions:

- The result has the attribute 'scenario' and is of type TestCaseResult.
- The result has the attribute 'why_needed' and is of type str.
- The result has the attribute 'key_assertions' and is a list of strings.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the ProviderContract handles an empty code by returning a valid result.

Why Needed: This test prevents a potential regression where an empty code would cause the contract to fail or return an incorrect result.

Key Assertions:

- The provider returns a non-empty result for the given test case.
- The result is not None, indicating that the contract handled the empty code correctly.
- The outcome of the annotation is 'passed', which is expected when handling an empty 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	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the `provider` object handles a `None` context for the `annotate` method.

Why Needed: This test prevents a potential bug where the `provider` object throws an exception or raises an error when handling a `None` context.

Key Assertions:

- The `provider` object is not None after calling `annotate` with `None` as the context.
- The `provider` object does not raise an exception or throw an error when handling a `None` context.
- The `provider` object correctly handles the `None` context by returning a valid result.
- The `provider` object does not crash or terminate unexpectedly when handling a `None` context.
- The `provider` object maintains its state and continues to function properly after handling a `None` context.
- The `provider` object returns a valid `TestCaseResult` object even when handling a `None` context.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: All providers should have an annotate method.

Why Needed: This test prevents a potential bug where some providers might not be able to annotate data.

Key Assertions:

- The provider has an attribute named 'annotate'.
- The provider's annotate method is callable.
- The provider's annotate method does not throw any errors when 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/llm/base.py	15 lines (ranges: 52-53, 245, 247, 249-250, 252-253, 255, 257-258, 260, 262-263, 265)
src/pytest_llm_report/llm/gemini.py	7 lines (ranges: 134, 136-139, 141-142)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The `annotate` method of the `GeminiProvider` class is called with a context that is too large, potentially causing performance issues or errors.

Why Needed: This test prevents a potential bug where the `annotate` method fails due to an excessive amount of data being passed in the context.

Key Assertions:

- The `annotate` method should not raise an exception when called with a context that is too large.
- The `annotate` method should only log warnings or errors for contexts that are too large, without raising exceptions.
- The number of annotations generated by the `annotate` method should be limited to prevent excessive memory usage in very large contexts.
- The `annotate` method should not raise an exception when called with a context that is too small (i.e., contains no data).
- The logging level for warnings or errors should be set to 'WARNING' or 'ERROR' instead of 'INFO'
- The `annotate` method should only log warnings or errors for contexts that are too large, without raising exceptions.
- The number of annotations generated by the `annotate` method should be limited to prevent excessive memory usage in very large contexts.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/base.py`	`9 lines (ranges: 52-53, 72, 75-76, 80, 165, 167, 175)`

src/pytest_llm_report/llm/gemini.py	155 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-221, 233, 245-248, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 346, 348-350, 352-355, 360-363, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417-418, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the provider does not properly handle cases where the required package is not installed, potentially leading to incorrect annotation or failure of tests.

Key Assertions:

- The annotation will contain an error message indicating that the required package is missing and how to install it.
- The annotation will include the correct path to install the required package.
- The annotation will report a clear and concise error message that does not require manual intervention.

COVERAGE

src/pytest_llm_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 a missing API token prevents the annotate method from working correctly.

Why Needed: The test verifies that the annotate method fails when an API token is not provided.

Key Assertions:

- The annotation should fail with an error message indicating that the GEMINI_API_TOKEN environment variable is not set.
- The annotation should check for the presence of the GEMINI_API_TOKEN environment variable before proceeding.
- The annotation should raise an exception when trying to access the API token without setting it as an environment variable.
- The annotation should provide a clear and descriptive error message indicating that the API token is missing.
- The annotation should not allow the test case to proceed with the annotated code even if the API token is set in the environment variables.
- The annotation should check for the presence of the GEMINI_API_TOKEN variable before attempting to access it.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/gemini.py	12 lines (ranges: 134, 136-139, 141-142, 160-161, 167-169)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that tokens are recorded correctly by the Gemini provider.

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

Key Assertions:

- The 'json' key in the captured dictionary contains the expected response data.
- The 'totalTokenCount' key in the captured dictionary matches the expected value.
- The 'candidates' list in the captured dictionary has at least one element with a valid content object.
- The 'usageMetadata' dictionary contains the expected rate limits.
- The 'tokenUsage' list in the 'limiter' dictionary has exactly one element with the expected token count.
- The 'tokenCount' value of the first element in the 'tokenUsage' list is equal to 123.
- The 'rateLimits' dictionary contains the expected rate limit for tokens per minute.

COVERAGE

src/pytest_llm_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 annotate retries on rate limits.

Key Assertions:

- The function `annotate_retries_on_rate_limit` should be called with an additional argument `rate_limit` when retrying.
- The annotation of retries on rate limits should be performed correctly.
- The error message or exception raised by the LLM provider for rate limit exceeded should include the correct reason.
- The LLM provider's retry logic should respect the specified rate limit.
- The test should fail if the `annotate_retries_on_rate_limit` function is not called with an additional argument `rate_limit` when retrying.
- The annotation of retries on rate limits should be performed correctly even if the error message or exception raised by the LLM provider does not include the correct reason.
- The LLM provider's retry logic should respect the specified rate limit even if the error message or exception raised by the LLM provider includes an incorrect reason.

COVERAGE

src/pytest_llm_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 LLM provider should rotate models on a daily limit to prevent model drift and ensure consistent performance.

Why Needed: This test prevents regression by ensuring that the LLM provider rotates models on a daily limit, which helps maintain consistency in model performance over time.

Key Assertions:

- The `rotate_models` method is called with a new list of models on each call to `annotate`.
- Each model in the new list has a unique ID.
- All models in the new list are from different datasets.
- No model is removed from the existing list of models.
- The number of models in the new list does not exceed the daily limit.
- The total number of models in the new list is equal to or less than the previous day's total.

COVERAGE

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

Key Assertions:

- The `annotate` method should not be called when the provider is on its daily limit.
- The `annotate` method should raise an error when the provider is on its daily limit.
- The `annotate` method should log a warning message when the provider is on its daily limit.
- The `annotate` method should update the provider's annotation status to 'skipped' when on daily limit.
- The `annotate` method should not call any other methods or functions that may be affected by the daily limit.
- The `annotate` method should not raise any exceptions when called from within a test.
- The provider's annotation status should be updated correctly after calling `annotate` with a daily limit.

COVERAGE

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

src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that LiteLLM provider annotates a valid response correctly.

Why Needed: Prevents regressions by ensuring the annotation is correct for valid responses.

Key Assertions:

- status ok
- redirect

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	177 lines (ranges: 39-42, 45-46, 48-49, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-101, 103, 105, 107-109, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-222, 225-227, 274-277, 280-283, 286, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-341, 343, 346, 348-350, 352-355, 360-366, 368-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function `test_exhausted_model_recovers_after_24h` is called with a valid input.
- The LLM model's `__call__()` method is called with a valid input.
- The model's `__call__()` method returns a valid result.
- The model's `__call__()` method does not raise an exception when exhausted.
- The function `test_exhausted_model_recovers_after_24h` checks if the model has recovered after 24 hours and verifies the correctness of its output.
- The LLM model's `__call__()` method is called with a valid input within 24 hours of the test call.
- The model's `__call__()` method returns a result that matches the expected value within 24 hours of the test 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	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 returns an error when no available models are found.

Why Needed: This test prevents a regression where the provider might return an error instead of raising a meaningful exception when no models are available.

Key Assertions:

- assertRaises(SystemExit) with message 'No available models found.'
- assertRaises(SystemExit) with message 'No available models found.' and raises ValueError
- assertRaises(SystemExit) with message 'No available models found.' and raises ValueError and isinstance(error, ValueError)
- assertRaises(SystemExit) with message 'No available models found.' and raises ValueError and isinstance(error, str)
- assertRaises(SystemExit) with message 'No available models found.' and raises ValueError and isinstance(error, Exception)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/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 should refresh after a specified interval.

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

Key Assertions:

- The `refresh_interval` attribute of the provider is set to the expected value.
- The `model_list` attribute is updated with the latest models within the specified time frame.
- The `refreshed_at` timestamp for each model is up-to-date and reflects the interval.
- No stale or outdated models are present in the list after the refresh.
- The provider's behavior remains consistent across different test runs.
- The `refresh_interval` attribute is correctly set to a reasonable value (e.g., 1 hour, 1 day).
- The `model_list` attribute contains only recent models within the interval.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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 regression where a completion error is not properly annotated by the provider.

Key Assertions:

- - The `error` attribute of the annotation contains the string 'boom', indicating a completion error.
- - The `completion` key in the annotation dictionary references the `fake_completion` function.
- - The ` LitellmProvider` instance is set to use the fake completion function.
- - A test case with a completion error is annotated by the provider correctly.
- - The annotation does not contain any other critical checks related to completion 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, 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 regression where the provider incorrectly handles invalid key_assertions payloads.

Key Assertions:

- Invalid response: key_assertions must be a list
- Missing or empty value for 'key_assertions' in 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/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 a missing dependency in the provided test case.

Why Needed: This test prevents a potential bug where the LiteLLM provider incorrectly reports a missing dependency when it is not installed.

Key Assertions:

- The annotation error message should include the correct path to install the 'litellm' package.
- The annotation error message should be in the format: 'package_name not installed. Install with: pip install package_name'.
- The annotation error message should be displayed when a test case is run without the required dependencies.
- The annotation error message should include the exact path to the missing dependency (in this case, 'litellm').
- The annotation error message should not contain any misleading or incorrect information about the installation 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/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 the `annotate` method of `LiteLLMProvider` returns an instance of `LlmAnnotation` with the expected attributes and assertions.

Why Needed: Prevents regressions by ensuring that the `annotate` method returns a valid annotation object.

Key Assertions:

- annotation is an instance of LlmAnnotation
- annotation.scenario matches 'Checks login'
- annotation.why_needed matches 'Stops regressions'
- annotation.key_assertions contains ['status ok', 'redirect']
- annotation.confidence is set to 0.8
- captured['model'] matches 'gpt-4o'
- captured['messages'][0]['role'] matches 'system'
- tests/test_auth.py::test_login is in captured['messages'][1]['content']
- def test_login() is in captured['messages'][1]['content'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/litellm_provider.py	20 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 66-67, 69-70)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the LiteLLM provider detects installed modules correctly.

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

Key Assertions:

- The `is_available()` method of the `LiteLLMProvider` instance should return True when the module is available.
- The `is_available()` method of the `LiteLLMProvider` instance should return False when the module is not available.
- The `is_available()` method of the `LiteLLMProvider` instance should raise an exception if the module is not found in the system modules.
- The `is_available()` method of the `LiteLLMProvider` instance should handle cases where the module has been removed from the system modules.
- The `is_available()` method of the `LiteLLMProvider` instance should correctly detect installed modules with different names or versions.
- The `is_available()` method of the `LiteLLMProvider` instance should not throw an exception if the module is not found in the system modules.

COVERAGE

src/pytest_llm_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: The test verifies that the annotate function handles context length errors correctly.

Why Needed: This test prevents a potential regression where the annotate function fails when the input context is too long.

Key Assertions:

- The annotate function should raise an error when the input context is too long.
- The annotate function should return False for a context that exceeds the default maximum length.
- The annotate function should not silently ignore errors and instead raise an exception with a meaningful message.
- The annotate function should provide a clear indication of why it failed (e.g. 'Context too long').
- The annotate function should handle cases where the input is already annotated (i.e. no context 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	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 annotation handles call errors correctly.

Why Needed: This test prevents regression where the annotation fails to handle call errors, potentially leading to false positives or incorrect results.

Key Assertions:

- The error message 'boom' is displayed when a call error occurs.
- The last error message 'boom' is displayed after 3 retries.
- The annotation correctly identifies the cause of the call error as 'boom'.
- The annotation does not display any other error messages for successful calls.
- The annotation displays an appropriate error message for system prompts.
- The annotation does not silently ignore or suppress call errors.
- The annotation provides a clear and concise description of the cause of the call error.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175)
src/pytest_llm_report/llm/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 the httpx dependency is missing.

Why Needed: This test prevents a potential bug where the provider incorrectly reports a non-existent issue.

Key Assertions:

- assert annotation.error == 'httpx not installed. Install with: pip install httpx'
- provider.annotate(test, 'def test_case(): assert True')
- test_case() should raise an error when httpx is missing

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 40-44)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_annotate_success_full_flow` 1ms 6

AI ASSESSMENT

Scenario: Test the full annotation flow of an Ollama provider with mocked HTTP responses.

Why Needed: Prevents authentication bugs by verifying that the response contains a valid token.

Key Assertions:

- Check if the status code is 200
- Validate the presence and format of the token in the response
- Verify that the token is not empty or None
- Check for any errors in the response (e.g., HTTP error codes)
- Ensure the token contains a specific key-value pair (in this case, 'scenario' and 'why_needed')

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/ollama.py	29 lines (ranges: 40-41, 47, 50, 52, 54-55, 57-60, 62-63, 114, 116-123, 127-130, 132, 134-135)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_call_ollama_success` 1ms 5

AI ASSESSMENT

Scenario: Test Ollama provider makes correct API call to generate a response.

Why Needed: Prevents regression in the Ollama provider's ability to make successful API calls.

Key Assertions:

- The 'url' captured is set to the expected URL for the API call.
- The 'json' captured contains the correct model and prompt data.
- The 'system' captured contains the correct system prompt data.
- The 'stream' captured is False, indicating no response stream was received.
- The 'timeout' captured matches the expected timeout value of 60 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	16 lines (ranges: 114, 116-123, 127-130, 132, 134-135)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_call_ollama_uses_default_model` 1ms 5

AI ASSESSMENT

Scenario: Test that Ollama provider uses default model when not specified.

Why Needed: Prevents regression where the Ollama provider does not use a default model.

Key Assertions:

- The `model` key in the captured JSON should be 'llama3.2'.
- The `model` value is set to an empty string, which is equivalent to using the default model.
- The `model` key is present in the captured JSON and its value matches the expected default model.
- No exception is raised when calling Ollama with a non-default model.
- The `model` key is not present in the captured JSON if it was not specified in the configuration.
- The `model` value is set to an empty string, which is equivalent to using the default model.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: The test verifies that the Ollama provider returns False when the server is unavailable.

Why Needed: This test prevents a regression where the provider incorrectly returns True when the server is available.

Key Assertions:

- The function _check_availability() of the OllamaProvider instance should return False when the server is not running.
- The connection error message 'Server not running' should be raised by the fake_get function.
- The provider's assertion that it can handle requests from a non-running server 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/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 provider incorrectly assumes all requests are successful (status code 200) when they may not be.

Key Assertions:

- assert provider._check_availability() == False
- assert isinstance(provider._check_availability(), bool)
- assert not provider._check_availability() in [True, None]
- assert provider._check_availability() != 200
- assert provider._check_availability() != 201
- assert provider._check_availability() != 202
- assert provider._check_availability() != 300
- assert provider._check_availability() != 400
- assert provider._check_availability() != 500

COVERAGE

src/pytest_llm_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: Verify that the Ollama provider checks availability via /api/tags endpoint successfully.

Why Needed: This test prevents a potential bug where the provider does not check for availability before making requests to the API.

Key Assertions:

- The `/api/tags` endpoint is present in the URL.
- The `provider._check_availability()` method returns True.
- A response with status code 200 is received from the `/api/tags` endpoint.
- The provider's `_check_availability()` method checks for availability before making requests to the API.
- The OllamaProvider instance has a valid configuration.
- The `ollama_host` attribute of the Config instance points to an available host.
- The provider uses the correct HTTPX client instance.
- No exceptions are raised when checking availability.

COVERAGE

src/pytest_llm_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 be considered local.

Why Needed: This test prevents a potential bug where the provider's location could be unknown or inconsistent.

Key Assertions:

- provider is an instance of OllamaProvider
- is_local() returns True for this 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	1 lines (ranges: 102)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_parse_response_invalid_json` 1ms 5

AI ASSESSMENT

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

Why Needed: This test prevents potential bugs where the Ollama provider incorrectly interprets valid JSON responses and reports incorrect errors.

Key Assertions:

- annotation.error == 'Failed to parse LLM response as JSON'
- provider._parse_response('not-json') is not None
- provider._parse_response('not-json').error != 'Invalid JSON'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 52-53, 186-187, 190-192)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-52, 55)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_parse_response_invalid_key_assertions` 1ms 5

AI ASSESSMENT

Scenario: The test verifies that the OllamaProvider rejects invalid key_assertions payloads in its _parse_response method.

Why Needed: This test prevents a potential bug where the provider incorrectly accepts or ignores invalid key_assertions payloads.

Key Assertions:

- The response data must contain a list of key_assertions
- Invalid key_assertion keys are not recognized by Ollama provider
- Missing key_assertions in response data raises an error

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	16 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207-209)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_parse_response_json_in_code_fence` 1ms 5

AI ASSESSMENT

Scenario: The provided test verifies that the Ollama provider correctly parses a JSON response from a markdown code fence.

Why Needed: This test prevents regression in case the JSON parsing logic changes, as it ensures the provider can extract the expected data even when the input is malformed or incomplete.

Key Assertions:

- The extracted JSON data should match the expected format.
- The `code_fence` property of the extracted JSON object should be set to the provided markdown code.
- The `annotations` property of the extracted JSON object should contain a list of strings, where each string represents an annotation.
- Each annotation in the `annotations` list should have a length between 1 and 100 characters.
- All annotations in the `annotations` list should be present in the provided markdown code.
- The `code_fence` property of each annotation object should match the original markdown code fence.
- Each annotation object's `annotations` property should contain exactly one string 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	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 function, `test_parse_response_json_in_plain_fence`, tests the functionality of extracting JSON from plain markdown fences without a specified language.

Why Needed: This test prevents regression in cases where Ollama providers are used with plain markdown fences that do not contain any language tags.

Key Assertions:

- The response should be a valid JSON object.
- The response should only contain string values (e.g., 'Hello, World!').
- The response should not contain any JSON objects or arrays.
- The response should not contain any properties that are not strings.
- The response should not contain any null or undefined values.
- The response should be a valid JSON object with the correct structure (i.e., object with string keys and values).
- The response should not have any circular references or nested objects.

COVERAGE

src/pytest_llm_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 Ollama provider parses valid JSON responses.

Why Needed: Prevents bugs in the LLM providers by ensuring they correctly parse and return valid JSON responses.

Key Assertions:

- assert a is not None
- assert b is not None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestArtifactEntry::test_to_dict` 1ms 3

AI ASSESSMENT

Scenario: Test that `CoverageEntry.to_dict()` correctly serializes the object.

Why Needed: Prevents regression in coverage entry serialization.

Key Assertions:

- The 'file_path' key is set to the expected value.
- The 'line_ranges' key is set to the expected format.
- The 'line_count' key is set to the expected value.
- A dictionary with the correct structure is returned.
- No KeyError is raised during serialization.
- The values of the keys are 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/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 instance.

Why Needed: This test prevents regression where the coverage data is not properly serialized to JSON.

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_data' key is not present in the dictionary and its value is an empty list.
- The 'file_path' key has a different value than the one passed to CoverageEntry.
- The 'line_ranges' key has a different value than the one passed to CoverageEntry.
- The 'line_count' key has a different value than the one passed to CoverageEntry.
- The 'coverage_data' key is not present in the dictionary and its value does not match any coverage data.
- The 'file_path' key is missing from the expected output.
- The 'line_ranges' key is missing from the expected output.
- The 'line_count' key is missing from the expected output.

COVERAGE

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

Why Needed: This test prevents a bug where the serialized CoverageEntry data is incorrect or incomplete.

Key Assertions:

- The 'file_path' key should match the expected value.
- The 'line_ranges' key should contain the correct ranges and values.
- The 'line_count' key should match the expected value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	4 lines (ranges: 40-43)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestLlmAnnotation::test_empty_annotation` 1ms 2

AI ASSESSMENT

Scenario: An empty annotation should be created with default values.

Why Needed: This test prevents a regression where an empty annotation would have no effect on the model's performance.

Key Assertions:

- annotation.scenario == "" (empty string)
- annotation.why_needed == "" (empty string) for LlmAnnotation class
- annotation.key_assertions == [] (no key assertions are performed in this annotation)
- assert annotation.confidence is None (default confidence value)
- assert annotation.error is None (default 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/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 all required fields.

Why Needed: This test prevents regression by ensuring that the minimal annotation is properly serialized and includes all necessary information.

Key Assertions:

- The presence of 'scenario' in the dictionary is expected.
- The presence of 'why_needed' in the dictionary is expected.
- The presence of 'key_assertions' in the dictionary is expected.
- The absence of 'confidence' in the dictionary is expected (as it is optional).

COVERAGE

src/pytest_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_dict_with_all_fields verifies that the full LlmAnnotation object is correctly converted to a dictionary.

Why Needed: This test prevents regression in LlmAnnotation conversion, ensuring consistency with expected output.

Key Assertions:

- Asserts that the 'scenario' field matches the specified value.
- Asserts that the 'confidence' field matches the expected value (0.95).
- Asserts that the 'context_summary' field has the correct mode ('minimal') and byte count (1000).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	10 lines (ranges: 104-107, 109-111, 113-115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestReportRoot::test_default_report 1ms 3

AI ASSESSMENT

Scenario: Test default report schema version and empty lists.

Why Needed: Prevents a regression where the default report does not have a valid schema version or contains non-empty lists of tests, warnings, or collection errors.

Key Assertions:

- d['schema_version'] == SCHEMA_VERSION
- d['tests'] == []
- not in d('warnings', 'collection_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	54 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestReportRoot::test_report_with_collection_errors` 1ms 3

AI ASSESSMENT

Scenario: Test Report with Collection Errors verifies that the test reports collection errors correctly.

Why Needed: This test prevents a bug where the report does not include all collection errors, potentially leading to incorrect reporting of issues.

Key Assertions:

- The length of `collection_errors` in the report should be 1.
- The value of `nodeid` in the first error in `collection_errors` should be 'test_bad.py'.
- All elements in `collection_errors` should have a non-empty `nodeid` attribute.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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 "Report with warnings" verifies that the ReportRoot class includes warnings in its report.

Why Needed: This test prevents a regression where the ReportRoot class does not include warnings in its reports.

Key Assertions:

- The length of the 'warnings' list should be 1.
- The code in the first warning should match 'W001'.
- The message of the first warning should match 'No coverage'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	60 lines (ranges: 229-231, 233, 235, 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestReportRoot::test_tests_sorted_by_nodeid 1ms 3

AI ASSESSMENT

Scenario: Tests should be sorted by nodeid in output.

Why Needed: This test prevents regression where the order of tests is not guaranteed to be consistent based on their nodeids.

Key Assertions:

- The list of nodeids returned from the `to_dict()` method of the ReportRoot object matches the expected order.
- The nodeids in the list are present in the original dictionary.
- The nodeids in the list do not contain any duplicates.
- The nodeids in the list are sorted in ascending order (nodeid values).
- The nodeids in the list are consistent across different runs of the test.
- No duplicate nodeids are present in the list.
- The nodeids in the list match the expected output for a given set of tests.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: The test verifies that the `to_dict` method of `ReportWarning` returns a dictionary with the 'detail' key containing the specified path.

Why Needed: This test prevents a warning about missing coverage information in the report.

Key Assertions:

- The 'detail' key should contain the specified path '/path/to/file'.
- The value of the 'detail' key should be exactly '/path/to/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/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 excluding detailed warnings in the report.

Key Assertions:

- The 'detail' key is expected to be present in the dictionary.
- The value of 'detail' should not be included in the output.
- The message and code should remain unchanged.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	5 lines (ranges: 229-231, 233, 235)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestRunMeta::test_aggregation_fields_present 1ms 3

AI ASSESSMENT

Scenario: Test RunMeta to ensure it has aggregation fields.

Why Needed: This test prevents regression where the aggregation policy is set to 'merge' without any source reports.

Key Assertions:

- The run_id field should be present in the meta data.
- The run_group_id field should be present in the meta data.
- 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 number of source reports 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 that LLM fields are excluded when annotations are disabled.

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

Key Assertions:

- The 'llm_annotations_enabled' key should not exist in the data.
- The 'llm_provider' key should not exist in the data.
- The 'llm_model' key should not exist in the data.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestRunMeta::test_llm_traceability_fields 1ms 3

AI ASSESSMENT

Scenario: Test LLM traceability fields are included when enabled.

Why Needed: This test prevents regression in the case where LLM traceability is disabled.

Key Assertions:

- The value of llm_annotations_enabled should be True.
- llm_provider should be set to 'ollama'.
- llm_model should be set to 'llama3.2:1b'.
- llm_context_mode should be set to 'complete'.
- llm_annotations_count should be an integer greater than 0.
- llm_annotations_errors should be an integer between 0 and 5.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 'Non-aggregated report should not include source_reports' verifies that a non-aggregated run does not include source reports.

Why Needed: This test prevents regression where the 'source_reports' key was included in aggregated runs, potentially masking important information.

Key Assertions:

- The 'source_reports' key is not present in the dictionary representation of the RunMeta object.
- The value of 'is_aggregated' is set to False for non-aggregated RunMeta objects.
- The presence of 'source_reports' in the dictionary representation indicates an aggregated run.
- Including 'source_reports' would mask important information about the source reports in a non-aggregated run.
- This test ensures that the correct behavior is observed when running without aggregation.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestRunMeta::test_run_meta_to_dict_full` 1ms 3

AI ASSESSMENT

Scenario: Test RunMeta to dict with all optional fields.

Why Needed: Prevents regression of missing or incorrect metadata in test results.

Key Assertions:

- Verify that the 'git_sha' field is set correctly.
- Verify that the 'git_dirty' field is set to True.
- Verify that the 'repo_version' field is set correctly.
- Verify that the 'repo_git_sha' field is set correctly.
- Verify that the 'repo_git_dirty' field is set to False.
- Verify that the 'plugin_git_sha' field is set correctly.
- Verify that the 'plugin_git_dirty' field is set to False.
- Verify that the 'config_hash' field is set correctly.
- Verify that the length of 'source_reports' is 1 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/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 RunMeta to include run status fields.

Why Needed: To prevent a potential bug where the 'run_status' field is not included in the output of the `to_dict()` method, which could lead to incorrect data being returned for certain test cases.

Key Assertions:

- The value of the 'exit_code' key should be 1.
- The value of the 'interrupted' key should be True.
- The value of the 'collect_only' key should be True.
- The value of the 'collected_count' key should be 10.
- The value of the 'selected_count' key should be 8.
- The value of the 'deselected_count' key 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	29 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestSchemaVersion::test_schema_version_format 1ms 2

AI ASSESSMENT

Scenario: Verifies the schema version is formatted as a semver string.

Why Needed: Prevents a potential bug where the schema version is not correctly parsed into its components.

Key Assertions:

- The `SCHEMA_VERSION` variable should be split into three parts using '.' as the separator.
- Each part of the `SCHEMA_VERSION` string should consist only of digits.
- If any part of the `SCHEMA_VERSION` string contains non-digit characters, it should 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/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: Verify that the `ReportRoot` class includes the schema version in its report root.

Why Needed: This test prevents a potential bug where the schema version is not included in the report root, potentially causing issues with data validation or reporting.

Key Assertions:

- The `schema_version` attribute of the `ReportRoot` instance should be equal to `SCHEMA_VERSION`.
- The `to_dict()` method of the `ReportRoot` instance should return a dictionary with a key named `schema_version` and value equal to `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: CoverageEntry should serialize correctly.

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 matches the expected value.
- The 'line_ranges' key in the dictionary matches the expected value.
- The 'line_count' key in the dictionary matches the expected value.
- The coverage entry data is correctly formatted and can be 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	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 missing some critical information.

Key Assertions:

- The 'scenario' key should be present in the returned dictionary.
- The 'why_needed' key should be present in the returned dictionary.
- The 'key_assertions' key should be present in the returned dictionary.
- The 'confidence' key should not be present in the returned 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: Verify that the `SourceReport` object includes its `run_id` in the dictionary representation.

Why Needed: This test prevents a bug where the `SourceReport` object is not properly serialized with its `run_id`.

Key Assertions:

- The `run_id` key should be present in the dictionary representation of the `SourceReport` object.
- The value of the `run_id` key should match the provided `run_id` parameter.
- The `run_id` key should not be empty or None.
- The `run_id` key should have a length of 10 characters (as specified in the `sha256` field).
- The `run_id` value should start with 'run-', followed by one or more alphanumeric characters, and then '-1'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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 that `CoverageEntry.to_dict()` correctly serializes the test summary.

Why Needed: This test prevents a potential bug where the serialized test summary is incorrect or incomplete, potentially leading to issues with coverage analysis.

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.
- Each required field ('file_path', 'line_ranges', and 'line_count') should be present in the dictionary.
- Any additional fields (e.g., 'start_line', 'end_line', etc.) should not be present in the dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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 a minimal result has the required fields.

Why Needed: This test prevents regression where a minimal result is missing required fields.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test verifies that the `TestCaseResult` object includes a coverage list.

Why Needed: This test prevents regression by ensuring that the coverage information is included in the result.

Key Assertions:

- The length of the 'coverage' key in the result dictionary should be 1.
- The value of the 'file_path' key in the first element of the 'coverage' list should match '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 that the `TestCaseResult` includes a flag when LLM opt-out is enabled.

Why Needed: This test prevents regression where LLM opt-out is not correctly handled in previous versions of the code.

Key Assertions:

- The value of `llm_opt_out` in the `result.to_dict()` method should be `True` when `llm_opt_out=True`.
- The key 'llm_opt_out' should exist in the dictionary returned by `result.to_dict()`.
- The value of `llm_opt_out` should match the expected value of `True`.
- When LLM opt-out is enabled, the result should not include any additional information.
- The presence of an additional key-value pair with a string 'LLM OPT OUT' in the dictionary returned by `result.to_dict()` is allowed.
- When LLM opt-out is enabled, the value of `llm_opt_out` should be `True`.
- The result should not include any other information that would indicate an error or exception occurred.
- The presence of a string 'LLM OPT OUT' in the dictionary returned by `result.to_dict()` indicates that LLM opt-out was 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/models.py	18 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180-182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test 'test_result_with_rerun' verifies that the rerun fields are included in the TestCaseResult.

Why Needed: This test prevents regression where a result is not properly reported with reruns.

Key Assertions:

- The value of `rerun_count` should be equal to 2.
- The value of `final_outcome` should be 'passed'.
- The `rerun_count` and `final_outcome` fields should be present in the TestCaseResult dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	19 lines (ranges: 161-165, 167, 169, 171, 173-176, 178, 180, 182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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

Why Needed: This test prevents regression by ensuring that rerun fields are excluded from the result dictionary.

Key Assertions:

- The 'rerun_count' field should not be present in the result dictionary.
- The 'final_outcome' field should not be present in the result dictionary.
- The 'rerun_fields' list should not contain any items that match 'rerun_count' or 'final_outcome'.
- The 'rerun_fields' list should only contain rerun fields (e.g., 'rerun_count', 'rerun_time', 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/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 potential regression where the default configuration may not be set properly, potentially leading to unexpected behavior or errors.

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: Verifies that the default configuration is properly returned by `get_default_config`.

Why Needed: 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'`.
- The assertion `cfg.provider == 'none'` checks if the `provider` attribute matches the expected value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test the `is_llm_enabled` check for different providers.

Why Needed: Prevents regression in case of provider changes or updates to the LLM model.

Key Assertions:

- The function should return False when the provider is set to 'none'.
- The function should return True when the provider is set to 'ollama' and the LLM is enabled.
- The function should not return a valid configuration object when the provider is set to 'ollama'.

COVERAGE

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

Scenario: The test verifies that the `validate()` method returns a list of error messages when an invalid aggregation policy is provided.

Why Needed: This test prevents a potential bug where the `validate()` method does not raise an exception when an invalid aggregate policy is passed, but instead returns an empty list or silently ignores errors.

Key Assertions:

- The first error message should contain 'Invalid aggregate_policy 'random''
- The error message should be present in the list of errors returned by `validate()`
- The test should fail when an invalid aggregation policy is provided, but no error messages are 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/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: Testing the `validate()` method of the `Config` class with an invalid provider.

Why Needed: Prevents a potential bug where an invalid provider is not properly validated, potentially leading to incorrect error messages or unexpected behavior.

Key Assertions:

- The `validate()` method should return exactly one error message.
- The first error message should contain the string 'Invalid provider '
- The error message should be present in the list of errors returned by `errors[0]`.
- Each error message should have a non-empty error message.
- The error messages should not be empty strings.
- The error messages should not be empty phrases (e.g., 'Invalid provider' instead of just 'invalid_provider'.)
- The error messages should not contain any whitespace characters except for one or two spaces between words.
- The error messages should not contain any special characters other than underscores and hyphens.

COVERAGE

src/pytest_llm_report/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 TestConfig is configured with invalid numeric ranges.

Key Assertions:

- cfg.validate() returns an error message indicating that llm_context_bytes must be at least 1000
- cfg.validate() returns an error message indicating that llm_max_tests must be 0 (no limit) or positive
- cfg.validate() returns an error message indicating that llm_requests_per_minute must be at least 1
- cfg.validate() returns an error message indicating that llm_timeout_seconds must be at least 1
- cfg.validate() returns an error message indicating that llm_max_retries must be 0 or positive

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	22 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209-218, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Valid configuration is validated without any errors.

Why Needed: This test prevents potential bugs that could occur when validating a valid configuration, ensuring the function does not return an error for well-formed input.

Key Assertions:

- The `validate()` method of the `Config` class returns an empty list (`[]`) if the input is valid.
- No exceptions are raised during validation when a valid configuration is provided.
- The function correctly handles invalid or missing required parameters 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/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 the ability to load aggregation options correctly.

Why Needed: This test prevents a regression where the aggregate policy and run ID are not being loaded correctly.

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'.

COVERAGE

src/pytest_llm_report/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 that the test_load_config_invalid_int_ini function handles invalid integer values in INI correctly.

Why Needed: This test prevents a potential regression where the test_load_config_invalid_int_ini function crashes or returns incorrect results when it encounters an invalid integer value in the INI file.

Key Assertions:

- The default value for llm_max_retries is set to 3.
- If the 'llm_report_max_retries' key contains an invalid integer value, the test should not crash or return incorrect results.
- The function should be able to handle invalid integer values without crashing or returning unexpected results.

COVERAGE

src/pytest_llm_report/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: Test loads coverage source option correctly.

Why Needed: This test prevents a bug where the coverage source is not set to 'cov_dir' when loading configuration.

Key Assertions:

- The value of `llm_coverage_source` in the loaded configuration is indeed 'cov_dir'.
- The `mock_pytest_config.option.llm_coverage_source` attribute is correctly set to 'cov_dir'.
- The coverage source option is properly applied when loading the configuration.
- The test does not fail when the coverage source is not set to 'cov_dir'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	25 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestLoadConfig::test_load_defaults` 1ms 3

AI ASSESSMENT

Scenario: Test that the default provider and report HTML are correctly loaded when no options are provided.

Why Needed: This test prevents a potential bug where the default provider or report HTML is not set, causing unexpected behavior in subsequent tests.

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: This test prevents a regression where the CLI overrides ini settings, potentially causing unexpected behavior or incorrect results.

Key Assertions:

- mock_pytest_config.getini.side_effect should be set to a lambda function that returns 'ini_value' for llm_report_html key and None otherwise
- mock_pytest_config.option.llm_report_html should be set to 'cli_report.html'
- mock_pytest_config.option.llm_requests_per_minute should be set to 100

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	27 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259-261, 263, 270-272, 274, 276, 278, 280-282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_options.py::TestLoadConfig::test_load_from_cli_retries 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `llm_max_retries` option is correctly set to 9 when loading configuration from CLI.

Why Needed: This test prevents a potential bug where the `llm_max_retries` option is not being updated correctly after setting it to a high value in the command line.

Key Assertions:

- The `llm_max_retries` option should be set to 9 when loading configuration from CLI.
- The `load_config()` function should update the `llm_max_retries` option with the new value.
- The `cfg.llm_max_retries` attribute should have the correct value after updating the option.
- The test should fail if the `llm_max_retries` option is not set to 9 when loading configuration from CLI.
- The `load_config()` function should raise an error if it cannot update the `llm_max_retries` option.
- The `cfg.llm_max_retries` attribute should be updated correctly after updating the option in the configuration.

COVERAGE

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

Why Needed: Prevents a potential bug where the report provider is not set correctly when loading config from ini file.

Key Assertions:

- The `provider` key in the configuration should be set to 'ollama'.
- The `model` key in the configuration should be set to 'llama3'.
- The `context_mode` key in the configuration should be set to 'balanced'.
- The `requests_per_minute` key in the configuration should be set to 10.
- The `max_retries` key in the configuration should be set to 5.
- The `html` key in the configuration should be set to 'report.html'.
- The `json` key in the configuration 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: Verify that the `aggregate_dir` is set correctly and matches the expected value.

Why Needed: This test prevents a potential bug where the aggregation settings are not properly configured, leading to incorrect reporting or analysis results.

Key Assertions:

- The `aggregate_dir` attribute of the `Config` object should be set to `/reports`.
- The `aggregate_policy` attribute of the `Config` object should be set to 'merge'.
- The `aggregate_include_history` attribute of the `Config` object should be set to True.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test Config with all output paths.

Why Needed: Prevents a potential bug where the test fails if any of the report formats are not set.

Key Assertions:

- The `report_html` is set to 'report.html'.
- The `report_json` is set to 'report.json'.
- The `report_pdf` is set to 'report.pdf'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the `capture_failed_output` option.

Why Needed: Prevents a potential issue where the test fails due to an unexpected output size.

Key Assertions:

- config.capture_failed_output should be set to `True`.
- config.capture_output_max_chars should not exceed 8000 characters.
- The captured output should not exceed 8000 characters in 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/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: Test the configuration of compliance settings.

Why Needed: This test prevents a bug where the configuration is not set correctly, potentially leading to security vulnerabilities.

Key Assertions:

- The `metadata_file` attribute of the `Config` object is set to 'metadata.json'.
- The `hmac_key_file` attribute of the `Config` object is set to 'key.txt'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test configuration with coverage settings.

Why Needed: Prevents a regression where the test is unable to determine which tests should be covered by the configuration.

Key Assertions:

- config.omit_tests_from_coverage is False
- config.include_phase == 'all'
- asserts that omit_tests_from_coverage is False
- asserts that include_phase is equal to 'all'

COVERAGE

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

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

Key Assertions:

- The '*.pyc' glob should be included in the config.
- The '*.log' glob should be included in the config.
- The custom exclude globs should override any default exclude globs.
- The custom exclude globs should not include any files that are intended to be processed by the LLM.
- The custom exclude globs should not include any directories that contain executable files.
- The custom exclude globs should only include files and directories that are specifically excluded from processing.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the `llm_context_include_globs` configuration option includes only `.py` files.

Why Needed: This test prevents a potential issue where the include globs are not correctly configured, potentially leading to incorrect or incomplete model training.

Key Assertions:

- The string `*.py` is in the list of include globs.
- The string `*.pyi` is in the list of include globs.
- The file `.py` is included in the list of include globs.
- The file `.pyi` is included in the list of include globs.
- The directory containing the files `.py` and `.pyi` is correctly identified as an include glob.
- The configuration option `llm_context_include_globs` is set to a valid string.
- The configuration option `llm_context_include_globs` does not contain any invalid characters or special regex patterns.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that `include_pytest_invocation` is set to `False` for the specified configuration.

Why Needed: This test prevents a potential bug where `include_pytest_invocation` is incorrectly set to `True` due to an incorrect pattern match in one of the invocation redact patterns.

Key Assertions:

- config.include_pytest_invocation should be set to `False`
- config.invocation_redact_patterns[0].match() should return `None` for the specified pattern
- config.invocation_redact_patterns[1].match() should return `None` for the specified pattern
- config.invocation_redact_patterns[2].match() should return `None` for the specified pattern

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 107)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify the correctness of LLM execution settings configuration.

Why Needed: Prevents a potential bug where the maximum number of tests is not set correctly, potentially leading to unexpected behavior or errors in test execution.

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.

COVERAGE

`src/pytest_llm_report/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 parameters exceeds 200.

Key Assertions:

- config.llm_include_param_values is True
- config.llm_param_value_max_chars == 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 regression in LLM settings when using the ollama provider.

Key Assertions:

- The `provider` attribute is set to 'ollama'.
- The `model` attribute is set to 'llama3.2'.
- The value of `llm_context_bytes` is 64000 bytes.
- The limit for the number of context files is 20.

COVERAGE

src/pytest_llm_report/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 a `Config` object is set to the specified path.

Why Needed: This test prevents a potential bug where the `repo_root` attribute is not correctly initialized with the expected path.

Key Assertions:

- The `repo_root` attribute of the `Config` object should be equal to the specified path (`/project`).
- The `Path` object passed to the `repo_root` parameter has the correct path (`/project`).
- The `config.repo_root` attribute is set to the specified value (`/project`).
- The `repo_root` attribute of the `Config` object does not contain any trailing slashes.
- The `repo_root` attribute of the `Config` object does not start with a slash.
- The `repo_root` attribute of the `Config` object is not an absolute path.
- The `repo_root` attribute of the `Config` object is set to a relative path (`./project` or `..`).
- The `config.repo_root` attribute is set to a non-existent directory (`/non_existent_project`).

COVERAGE

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

Why Needed: This test prevents a potential bug where invalid or missing include_phase values are passed into the Config class, potentially causing validation errors or unexpected behavior.

Key Assertions:

- The validate() method of the Config object should not return any error messages for valid include_phase values.
- All include_phase values in the config.validate() output should be empty strings.
- Any include_phase value that is missing from the config.validate() output should be ignored and not reported as an error.
- If a phase value is present but invalid, it should be ignored and not reported as an error.
- The validate() method should only report errors for include_phase values that are explicitly set in the Config object.
- Any include_phase value that is explicitly set to False or None should be ignored and not reported as an error.
- If a phase value is present but missing from the config, it should be ignored and not reported as an error.
- The validate() method should only report errors for include_phase values that are explicitly set in the Config object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Verify that the default exclude globs are correctly set.

Why Needed: This test prevents a potential bug where the default exclude globs do not match the expected values, potentially leading to unexpected behavior or errors.

Key Assertions:

- The string `*.pyc` is present in the defaults.
- The string `__pycache__/*` is present in the defaults.
- The string `*secret*` is present in the defaults.
- The string `*password*` is present in the defaults.

COVERAGE

src/pytest_llm_report/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: Verify the presence of default redact patterns in the test configuration.

Why Needed: This test prevents a potential security vulnerability where sensitive information like passwords and tokens are not properly redacted.

Key Assertions:

- The `--password` pattern should be present in the test configuration.
- The `--token` pattern should be present in the test configuration.
- The `--api[_-]?key` pattern should be present in the test configuration.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test default values of TestConfigDefaultsMaximal.

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

Key Assertions:

- config.provider should be 'none'
- config.llm_context_mode should be 'minimal'
- config.llm_context_bytes should be 32000
- config.omit_tests_from_coverage should be True
- config.include_phase should be '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 regression in case a provider's configuration changes or is updated.

Key Assertions:

- The method should return False when the provider is set to 'none'.
- The method should return True when the provider is set to 'ollama', 'litellm', or 'gemini'.

COVERAGE

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

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_aggregate_policy` 1ms 3

AI ASSESSMENT

Scenario: Test the validation of an invalid aggregate policy.

Why Needed: To prevent a potential bug where an invalid aggregate policy is passed to the Config class, causing unexpected behavior or errors.

Key Assertions:

- The validate method should return exactly one error for an invalid aggregate policy.
- 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

Scenario: Test validates the maximum options configuration with an invalid context mode.

Why Needed: This test prevents a potential bug where the validation of the maximum options configuration returns incorrect results due to an invalid context mode.

Key Assertions:

- The function `Config(llm_context_mode='invalid')` should raise an error when given an invalid context mode.
- The error message for the invalid context mode should include 'Invalid llm_context_mode 'invalid''.
- At least one error should be returned from the validation process.
- The error messages should contain information about the invalid context mode.
- The test should fail if no errors are found during the validation process.
- The function `Config(llm_context_mode='invalid')` should not return an empty list of errors when given a valid context mode.
- The error message for a valid context mode should be different from that of an invalid context mode.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-189, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 fails due to an invalid include phase.

Key Assertions:

- The config should return exactly one error message for an invalid include phase.
- The error message should contain 'Invalid include_phase 'invalid''.
- The invalid include phase should be present in the error 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/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

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	19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_numeric_bounds` 1ms 3

AI ASSESSMENT

Scenario:
tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_numeric_bounds

Why Needed: Prevents regression where the llm_context_bytes, llm_max_tests, llm_requests_per_minute, or llm_timeout_seconds are set to invalid numeric values.

Key Assertions:

- The validate method should return an error for any invalid numeric value in config.
- config should contain errors if llm_context_bytes is not a non-negative integer.
- config should contain errors if llm_max_tests is not between 0 and 10 (inclusive).
- config should contain errors if llm_requests_per_minute is not between 0 and 100 (inclusive).
- config should contain errors if llm_timeout_seconds is not between 1 and 60 (inclusive).

COVERAGE

src/pytest_llm_report/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 an invalid configuration is validated correctly.

Why Needed: Prevents a potential bug where an invalid configuration would be silently ignored and cause unexpected behavior in the application.

Key Assertions:

- The `validate()` method of the Config object should return an empty list for a valid configuration.
- An error message or indication that the configuration is invalid should be provided by the `validate()` method.
- The `validate()` method should not silently ignore the input configuration and instead report an error.

COVERAGE

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

Why Needed: Prevents a potential bug where the config is missing safe defaults.

Key Assertions:

- The function `load_config(pytestconfig)` returns an instance of `Config`.
- The assertion `assert isinstance(cfg, Config)` checks if the returned value is indeed a `Config` object.
- The assertion `cfg = load_config(pytestconfig)` assigns the result to the variable `cfg` and then checks its type using `isinstance()`.
- If pytest's options are not registered, `pytestconfig` will be `None`, and attempting to assign it to `cfg` would raise an error.
- The assertion `assert cfg is not None` ensures that the config was successfully loaded without raising an exception.
- The function `load_config()` likely checks for specific configuration settings or options before returning a default value.
- If pytest's options are missing, `load_config()` might return an empty object or raise an error, which would be caught by this assertion.
- This test verifies that the config defaults are correctly set and can be loaded without 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	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 and not None.

Why Needed: Prevent a potential bug where the plugin configuration is inaccessible due to a missing or incorrect `pytestconfig` setup.

Key Assertions:

- The `pytestconfig` object should be an instance of `pytest.config.Config`.
- The `pytestconfig` object should not be None.
- The `pytestconfig` object should have attributes that are accessible (e.g. `add_markers`, `markers`, 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/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 process.

Key Assertions:

- The `test_llm_context_marker` function should return `True` without any errors.
- The `src/pytest_llm_report/collector.py` module should not raise any exceptions when calling the `handle_collection_report` method.
- The `src/pytest_llm_report/plugin.py` module should not raise any exceptions when registering itself as a pytest plugin.
- The `pytest_addoption` function from `pytest` should add the required command-line options for the `llm-report` plugin without raising any errors.
- The `handle_collection_report` method in `src/pytest_llm_report/plugin.py` should not raise any exceptions when collecting test results.

COVERAGE

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

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: The `requirement_marker` function is being tested to ensure it does not throw any errors when used.

Why Needed: This test prevents a potential bug that could cause unexpected behavior or errors in the plugin integration process.

Key Assertions:

- The `requirement_marker` function should be called without causing any exceptions or errors.
- Any attempts to call `requirement_marker` should result in no output or an empty string.
- If an exception is raised, it should not propagate up the call stack and cause the test to fail.
- The `requirement_marker` function should not modify any external state or variables.
- It should be possible to call `requirement_marker` multiple times without causing any issues.
- Any side effects caused by calling `requirement_marker` should not affect other parts of the plugin integration code.
- If an error occurs while calling `requirement_marker`, it should be handled properly and not cause the test to 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/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_plugin_integration.py::TestReportGeneration::test_report_writer_integration` 34ms 6

AI ASSESSMENT

Scenario: The test verifies that the report writer correctly generates a full report with both JSON and HTML formats.

Why Needed: This test prevents regression or bug in the report writer's functionality, ensuring consistency in the output of reports generated by pytest_llm_report.

Key Assertions:

- The total number of tests passed should be equal to the total number of tests run.
- The 'passed' count for each test should match the actual number of tests that ran successfully.
- At least one test failed and its error message should be present in the report.
- All test nodes should have a unique node ID.
- Each test result should contain an outcome ('passed', 'failed') and a duration (in seconds).
- The summary section should include both total and passed counts for all tests.
- The HTML report should contain references to at least two different Python files (test_a.py and test_b.py).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	79 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 collectreport skips when disabled and pytest_collectreport is mocked correctly.

Why Needed: This test prevents a regression where collectreport fails to run or produces incorrect results when pytest_collectreport is not enabled.

Key Assertions:

- The `pytest_collectreport` function should be called with the `_enabled_key` as an argument and its return value should be `False`.
- The `get` method of `session.config.stash` should be called with `_enabled_key` as an argument and its return value should be `None`.
- The `pytest_collectreport` function should not have been called when the mock report is created.
- The `get` method of `session.config.stash` should not have been called before setting it to `False`.
- The `pytest_collectreport` function should not have been called with a non-boolean value as an argument.
- The `get` method of `session.config.stash` should return `None` when the `_enabled_key` is not found in the stash.
- The `pytest_collectreport` function should raise an exception when it cannot find the `_enabled_key` in the stash.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Verify that collectreport calls collector when enable is true.

Why Needed: This test prevents a potential bug where the collectreport function does not call the collector even if it is enabled.

Key Assertions:

- mock_collector.handle_collection_report.assert_called_once_with(mock_report)
- mock_report.session.config.stash.get.mocked()._enabled_key == True
- mock_report.session.config.stash.get.mocked()._collector_key == mock_collector

COVERAGE

src/pytest_llm_report/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` attribute is missing from the report.

Why Needed: Prevents a potential bug where `pytest_collectreport` throws an error due to missing session information.

Key Assertions:

- The function `pytest_collectreport` should not raise an exception when it encounters a mock report without a `session` attribute.
- A `mock_report` object with no `session` attribute is expected to pass the test.
- No error message or indication of failure is expected in this 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/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: Verifies that the `pytest_collectreport` function does not throw an exception when a `None` session is provided.

Why Needed: Prevents a potential bug where the plugin throws an error due to a missing or invalid session.

Key Assertions:

- The `pytest_collectreport` function should not raise an exception when given a `None` session.
- The `session` attribute of the mock report object is set to `None` before calling `pytest_collectreport`.
- The `pytest_collectreport` function does not throw any errors or exceptions when called with a `None` session.

COVERAGE

`src/pytest_llm_report/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 regression where the LLM report configuration is not properly handled, potentially leading to unexpected behavior or errors.

Key Assertions:

- mocks.pytest_configure() was called with the correct mock_config instance.
- The 'llm_report_provider' key in the ini_values dictionary matches the expected value.
- The value of 'llm_report_html', 'llm_report_json', and 'llm_report_pdf' keys are None as expected.
- The value of 'llm_evidence_bundle' key is None as expected.
- The value of 'llm_dependency_snapshot' key is None as expected.
- The value of 'llm_requests_per_minute' key is None as expected.
- The value of 'llm_aggregate_dir', 'llm_aggregate_policy', and 'llm_aggregate_run_id' keys are None as expected.
- The value of 'llm_max_retries' key matches the expected value.
- mocks.pytest_configure() did not raise a UserWarning with the correct 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	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 setting invalid configuration options.

Why Needed: Prevents regression where the plugin fails to configure due to invalid or missing required parameters.

Key Assertions:

- The `pytest_configure` function should raise a `UsageError` with a message indicating configuration errors.
- The `pytest_configure` function should not accept any arguments other than `mock_config` and `mock_config.option.*`.
- The `pytest_configure` function should set the `llm_report_html`, `llm_report_json`, `llm_report_pdf`, `llm_evidence_bundle`, `llm_dependency_snapshot`, `llm_requests_per_minute`, `llm_aggregate_dir`, `llm_aggregate_policy`, `llm_aggregate_run_id`, and `llm_aggregate_group_id` options to None.
- The `pytest_configure` function should set the `rootpath` option to `/project`.
- The `pytest_configure` function should set the `stash` option to an empty dictionary.
- The `pytest_configure` function should not raise a `UsageError` when called with valid configuration options.
- The `pytest_configure` function should accept any arguments other than `mock_config` and `mock_config.option.*`.
- The `pytest_configure` function should set the `llm_report_provider`, `llm_report_html`, `llm_report_json`, `llm_report_pdf`, `llm_evidence_bundle`, `llm_dependency_snapshot`, `llm_requests_per_minute`, `llm_aggregate_dir`, `llm_aggregate_policy`, `llm_aggregate_run_id`, and `llm_aggregate_group_id` options to valid values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: Verify that configure skips on xdist workers when adding a marker.

Why Needed: Prevent regression in pytest llm_report plugin where skipping configuration due to xdist workers is desired but the addinivalue_line call is still made for markers before worker check.

Key Assertions:

- mock_config.addinivalue_line was not called.
- addinivalue_line is still called for markers before worker check.
- pytest_configure() is called with mock_config as argument but it does not skip configuration due to xdist workers.

COVERAGE

src/pytest_llm_report/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 plugin configuration failures due to missing Config.load method.

Key Assertions:

- mock_load.call_args[0].validate.return_value is None
- mock_load.call_args[1].load_config.return_value is mock_cfg
- pytest_configure.asserts_called_once() 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	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 llm_report plugin.

Why Needed: Prevents a potential bug where CLI options are not set, leading to incorrect configuration.

Key Assertions:

- The `llm_report_html` option is set to `None`.
- The `llm_report_json` option is set to `None`.
- The `llm_report_pdf` option is set to `None`.
- The `llm_evidence_bundle` option is set to `None`.
- The `llm_dependency_snapshot` option is set to `None`.
- The `llm_requests_per_minute` option is set to `None`.
- The `llm_aggregate_dir` option is set to `None`.
- The `llm_aggregate_policy` option is set to `None`.
- The `llm_aggregate_run_id` option is set to `None`.
- The `llm_aggregate_group_id` option is set to `None`.
- The `llm_max_retries` option is set to `None`.
- The `rootpath` option is set to `/project`.
- The `provider`, `model`, `llm_context_mode`, `requests_per_minute`, and `aggregate_dir` options are set 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	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 regression where the CLI overrides INI settings, potentially causing unexpected behavior in the plugin's configuration.

Key Assertions:

- The `llm_report_html` option is set to 'cli.html'.
- The `llm_report_json` option is set to 'cli.json'.
- The `report_pdf` option is set to 'cli.pdf'.
- The `report_evidence_bundle` option is set to 'bundle.zip'.
- The `report_dependency_snapshot` option is set to 'deps.json'.
- The `llm_requests_per_minute` option is set to 20.
- The `aggregate_dir` option is set to '/agg'.
- 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 `rootpath` option is 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 and stash returns False for enabled.

Why Needed: Prevents regression where the plugin's terminal summary is not checked correctly when it is enabled but stash returns a value indicating it should be skipped.

Key Assertions:

- mock_config.stash.get.assert_called_once_with(_enabled_key, False)
- assert result is None
- mock_config.workerinput is not 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	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 configured to skip workers.

Why Needed: This test prevents a regression where the plugin incorrectly processes terminal summaries for an xdist worker.

Key Assertions:

- The `pytest_terminal_summary` function should return None immediately without processing any results.
- The `workerinput` attribute of the mock configuration object is set to 'gw0' indicating that the xdist worker ID is 'gw0'.
- The test does not perform any assertions or checks on the returned value, ensuring it behaves as expected.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`8 lines (ranges: 238-239, 387-388, 391, 395-397)`

PASSED tests/test_plugin_maximal.py::TestPluginMaximal::testload_config 4ms 3

AI ASSESSMENT

Scenario: Test config loading from pytest objects (CLI + INI) to ensure the correct value is returned for `report_html` option.

Why Needed: This test prevents a potential bug where the `report_html` option is not set correctly in the loaded configuration.

Key Assertions:

- The `report_html` option should be set to 'out.html'.
- The value of `report_html` should match the expected string 'out.html'.
- The `getini` method should return None for the specified key, but the test does not verify this.
- The `rootpath` attribute should have been set to '/root' during configuration loading.
- The `llm_report_json` option should be set to 'out.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	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: The test prevents a regression where the plugin's report generation is not properly handled when makereport is disabled.

Key Assertions:

- mock_item.config.stash.get() returns False
- mock_call.return_value is None
- gen.send(mock_outcome) does not raise an exception
- mock_outcome.get_result().result is None
- next(gen) yields a StopIteration exception

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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: This test prevents a potential bug where the plugin does not collect reports even when makereport is enabled.

Key Assertions:

- The `pytest_runtest_makereport` function should call the `handle_runtest_logreport` method of the `mock_collector` instance.
- The `handle_runtest_logreport` method should be called with a `mock_report` object as its first argument and `mock_item` as its second argument.
- The `get_result` method of the `mock_outcome` object should return `True` when called on the `mock_collector` instance.
- The `handle_runtest_logreport` method should be called with a `mock_report` object as its first argument and `mock_item` as its second argument, even if it returns immediately without calling `get_result`.
- The `handle_runtest_logreport` method should not call `get_result` on the `mock_outcome` object when it is called before `get_result` has a chance to return.
- The `handle_runtest_logreport` method should be able to handle an empty `mock_report` object without raising an exception.
- The `handle_runtest_logreport` method should not call `get_result` on the `mock_outcome` object when it is called after `get_result` has returned a value.

COVERAGE

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

PASSED `tests/test_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 is called.

Why Needed: collection_finish is not executed when it's disabled in pytest.

Key Assertions:

- mock_session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_collection_finish(mock_session) should be called with a mock session that returns False for stash.get(_enabled_key, False)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 387-388, 391, 395-397, 431-432)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_enabled` 2ms 2

AI ASSESSMENT

Scenario: Verify that collection_finish calls collector when enabled in pytest.

Why Needed: This test prevents a potential regression where the collector is not called when collection_finish is enabled.

Key Assertions:

- The stash_get method returns True for _enabled_key and mock_collector.
- The stash_get method returns mock_collector for _collector_key.
- The pytest_collection_finish function calls mock_collector with mock_session.items.
- mock_collector.handle_collection_finish is called once with mock_session.items.
- The collection_finish call is made before the collector's first call to handle_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 checks the stash for enabled status.

Why Needed: To prevent a regression where pytest_sessionstart fails to check the stash when the plugin is disabled.

Key Assertions:

- mock_session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_sessionstart(mock_session) should not be called
- mock_session.config.stash.get.return_value should have been set to False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 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 created if pytest_sessionstart is disabled.

Key Assertions:

- The `_collector_key` should be present in `mock_stash`.
- The `_start_time_key` should be present in `mock_stash`.
- A mock session with a configured stash that supports both get() and [] methods should have the collector created.
- The collector should be accessible through `_collector_key` in `mock_stash`.
- The start time of the collection process should be recorded through `_start_time_key` in `mock_stash`.
- A mock session with a configured stash that supports both get() and [] methods should have its stash dictionary populated correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	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 add all expected arguments if the plugin is not properly configured or if there are other plugins interfering with its functionality.

Key Assertions:

- The 'llm-report' group should be added to the parser.
- The 'LLM-enhanced test reports' option should be recognized by pytest_addoption.
- The 'llm-coverage-source' group should be added to the parser if it is present in the options.
- Any other expected arguments should be correctly identified and added to the parser.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	99 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption_ini` 2ms 2

AI ASSESSMENT

Scenario: Test pytest_addoption adds INI options (lines 13-34) to ensure it correctly handles plugin configuration.

Why Needed: This test prevents a regression where pytest_addoption does not add INI options for plugins, potentially causing issues with plugin functionality.

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 calculation when using the `terminal_summary` feature.

Key Assertions:

- The `coverage_report` method should be called with a valid `CoverageMapper` instance.
- The `report_html` parameter should be set to a string value.
- The `stash` dictionary should contain both `_enabled_key` and `_config_key` keys.
- The `mock_cov_cls.return_value` attribute should match the expected coverage report.
- The `MockStash.stash` dictionary should not have any additional keys besides `_enabled_key` and `_config_key`.
- The `coverage.Coverage.load` method should be called with a valid `CoverageMapper` instance.
- The `pytest_llm_report.report_writer.ReportWriter` class should be instantiated correctly.

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: Tests terminal summary with LLM enabled.

Why Needed: Prevents regression in terminal summary functionality when LLM is enabled.

Key Assertions:

- Verify that the `pytest_terminal_summary_llm_enabled` test passes without any errors.
- Check if the `mock_annotate` function is called once with the correct configuration.
- Verify that the `stash` dictionary contains the expected keys (`_enabled_key` and `_config_key`) when LLM is enabled.
- Ensure that the `mock_terminalreporter.stats` dictionary does not contain any unexpected values.
- Verify that the `pytest_llm_report.llm.annotator.annotate_tests` function returns a tuple with the correct arguments (0, cfg).
- Check if the `mock_annotate.call_args[0][1] == cfg` assertion passes without any errors.

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 potential regression where the plugin does not create a collector even when it is supposed to be present in the configuration.

Key Assertions:

- The `pytest_terminal_summary` function should return an instance of `pytest_llm_report.plugin._terminal_summary.Collector` if `_enabled_key` is set to `True` and `_config_key` has a valid value.
- The `stash` attribute of the `_config_key` dictionary should be an instance of `pytest_llm_report.plugin._terminal_summary.Collector` if it was created with the correct configuration.
- The `coverage_map` attribute of the `_config_key` dictionary should be an empty dictionary if its mapping is not properly set up.
- If `_enabled_key` is set to `True`, the `pytest_terminal_summary` function should call `get()` on the `_stash` instance with a valid key.
- The `pytest_terminal_summary` function should return `None` if it cannot find any matching collector in the stash.
- The `pytest_terminal_summary` function should not raise an exception when creating a new collector without setting `_enabled_key` to `True` or `_config_key` having a valid value.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	45 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_with_aggregation` 3ms 3

AI ASSESSMENT

Scenario: Test terminal summary with aggregation enabled.

Why Needed: Prevents regression in aggregation functionality when using terminal summary.

Key Assertions:

- The `aggregate_dir` is set to `/agg` and the report directory is created successfully.
- The `get()` method of the stash returns a list of files, while the `[[]]` method returns an empty list.
- The `aggregate()` method of the aggregator returns a report object, which is then written to JSON and HTML files.
- The `ReportWriter` class writes JSON and HTML files successfully.

COVERAGE

src/pytest_llm_report/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.

Why Needed: This test prevents a regression where the plugin fails to calculate coverage percentage due to an OSError during load.

Key Assertions:

- The `load` method of `CoverageMapper` should not raise an exception when the coverage map is loaded successfully.
- The `coverage_map` attribute of the mocked `CoverageMapper` instance should be set to a valid Coverage object.
- The `report_writer` attribute of the mocked `ReportWriter` instance should be set to a valid ReportWriter instance.
- The `coverage` attribute of the mocked `Coverage` object returned by `load` should not be an OSError.
- The `coverage_map` attribute of the mocked `CoverageMapper` instance should have a valid coverage map.
- The `report_writer` attribute of the mocked `ReportWriter` instance should have a valid report writer.
- The `coverage` attribute of the mocked `Coverage` object returned by `load` should be set to a valid Coverage object.
- The `report_html` parameter of the `pytest_terminal_summary` function should not raise an exception when the coverage map is loaded successfully.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	52 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-305, 322-325, 331-332, 337-338, 365-375, 387-388, 391, 395-397)

PASSED `tests/test_prompts.py::TestContextAssembler::test_assemble_balanced_context` 8ms 4

AI ASSESSMENT

Scenario: Test the ContextAssembler to assemble a balanced context for a test file with dependencies.

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

Key Assertions:

- The 'utils.py' file is present in the assembled context.
- The 'def util()' function is found within the 'utils.py' file.
- All lines of the 'utils.py' file have a line count greater than zero.
- The 'utils.py' file has at least one line range that includes the start and end of the file.
- The assembled context does not include any files other than 'utils.py'.
- The assembled context includes all required functions from 'utils.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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	51 lines (ranges: 33, 49, 52, 55, 58, 60-61, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 116, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-155, 158-159, 163, 191-192, 194)

PASSED `tests/test_prompts.py::TestContextAssembler::test_assemble_complete_context` 1ms 4

AI ASSESSMENT

Scenario: Assembling a complete context for the 'test_a.py' test file.

Why Needed: Prevents regression when using the 'complete' llm_context_mode, which may cause issues with incomplete contexts.

Key Assertions:

- The 'source' variable will contain the actual source code of the 'test_1' function.
- The 'context' variable will be populated with a valid context for the 'test_a.py' test file.
- The 'source' and 'context' variables will not be empty after calling the 'assemble' 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)
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: Test that the ContextAssembler can assemble a minimal context for a test file with no dependencies.

Why Needed: This test prevents a potential bug where the ContextAssembler fails to assemble a minimal context when there are no dependencies in the test file.

Key Assertions:

- The 'test_1' function is present in the source code of the test file.
- The 'test_1' function has an outcome of 'passed'.
- The ContextAssembler successfully assembles a minimal 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	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: The test verifies that the ContextAssembler does not exceed the specified context limits when assembling a file with excessive content.

Why Needed: This test prevents potential memory issues caused by exceeding the LLM context limit or having too much code in the assembly result.

Key Assertions:

- The assembler should be able to assemble the given file without truncating it.
- The assembled context should include the specified file path.
- ... The assembled context should not exceed 40 bytes (20 + 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: Verify the ContextAssembler's ability to handle non-existent file names and nested test names with parameters.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly handles files that do not exist or have nested test names with parameters.

Key Assertions:

- assert 'def test_param' in source
- assert source.startswith('test_p.py::')
- assert source.split(':').index('test_param[1]') 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	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 excludes certain files from the context.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly includes files in the context.

Key Assertions:

- asserts that 'test.pyc' is excluded from the context because it's a compiled Python file.
- asserts that 'secret/key.txt' is excluded from the context because it's a secret file.
- asserts that 'public/readme.md' is not included in the context because it's public and shouldn't be accessed directly.

COVERAGE

src/pytest_llm_report/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 for consecutive lines.

Why Needed: This test prevents a regression where consecutive lines are not compressed correctly.

Key Assertions:

- assert compress_ranges([1, 2, 3]) == '1-3'
- the list `[1, 2, 3]` should be compressed to `'1-3'`
- the range `1-3` is correct

COVERAGE

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

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/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: The current implementation of `compress_ranges` does not handle empty lists correctly, which may lead to incorrect results or unexpected behavior.

Key Assertions:

- Input list is empty.
- Output should be an empty string.
- No compression occurs when the input list is empty.
- Error handling for non-compressible ranges is missing.
- Compression of empty range is not handled correctly.
- Test case covers edge case where input list is empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 'test_mixed_ranges' verifies that the `compress_ranges` function handles mixed ranges correctly.

Why Needed: This test prevents regression in case of mixed ranges where a single number is used as a range.

Key Assertions:

- The output should be '1-3, 5, 10-12, 15'.
- The output should contain all the numbers from the input list without any gaps.
- The output should not include any numbers outside the specified ranges.
- The function should handle single numbers correctly as a range.
- The function should preserve the original order of the input list.
- The function should not add any extra comma between the ranges.
- The function should return an empty string if the input list is empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Non-consecutive lines in the input list should be separated by commas.

Why Needed: This test prevents a potential bug where non-consecutive lines are not properly compressed.

Key Assertions:

- The function `compress_ranges([1, 3, 5])` should return a string with comma-separated integers.
- The integers in the input list should be separated by commas.
- Any non-integer values in the input list should be ignored during compression.
- The resulting compressed string should contain only integers.
- Non-consecutive integer pairs should be separated by commas.
- Compressed strings with consecutive integer pairs should not be returned.
- The function should handle 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	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 incorrectly handles single-line inputs without range notation.

Key Assertions:

- Input should be an array of integers.
- Input length should be greater than 0.
- Compressed value should match the original input.
- No error message or exception is thrown for a single line.
- Function does not raise an exception when given a non-array input.
- Function handles empty input correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	10 lines (ranges: 29, 33, 35-37, 39, 50, 52, 65-66)

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

AI ASSESSMENT

Scenario: Testing the 'two_consecutive' method in the `compress_ranges` function.

Why Needed: This test prevents regression when two consecutive numbers are compressed into a single range.

Key Assertions:

- The input list should contain exactly one element.
- The first element of the input list should be less than or equal to the second element.
- The difference between the elements in the input list should be greater than zero.
- The resulting compressed range should match '1-2'.
- If the input list contains more than two consecutive numbers, 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

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

AI ASSESSMENT

Scenario: Test 'test_unsorted_input' verifies that the function handles unsorted input correctly.

Why Needed: This test prevents a potential bug where the function would incorrectly group ranges in an unsorted input.

Key Assertions:

- The input list is sorted before passing it to `compress_ranges`.
- The output string contains the correct range groups (1-3 and 5).
- No other critical checks are performed by this 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/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: Testing the `expand_ranges` function with an empty string.

Why Needed: This test prevents a potential bug where an empty string is not handled correctly by the `expand_ranges` function.

Key Assertions:

- The `expand_ranges` function should return an empty list when given an empty string as input.
- An empty string should be considered as a valid range.
- The function should handle edge cases where the input string is empty or contains only whitespace characters.
- Any non-string values in the input string should be ignored and not included in the result.
- The function should raise an error if the input string is not a string.
- The function should return an empty list for an empty string.
- The function should handle strings with multiple consecutive whitespace characters correctly.
- Any non-string values in the input string should be ignored and not included in the result.

COVERAGE

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

Why Needed: This test prevents a potential bug where the function might incorrectly handle ranges with overlapping values.

Key Assertions:

- A single value is not included in the expanded range.
- Ranges are separated by commas and inclusive or exclusive endpoints are handled correctly.
- The function handles overlapping ranges correctly (e.g., `1-3` and `10-12`).
- Single values are correctly identified as individual numbers rather than part of a larger range.
- Inclusive endpoints are handled correctly (e.g., `1, 5, 10-12`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	11 lines (ranges: 81, 84-91, 93, 95)

PASSED tests/test_ranges.py::TestExpandRanges::test_range 1ms 3

AI ASSESSMENT

Scenario: The `expand_ranges` function is expected to correctly expand a single range.

Why Needed: This test prevents the function from expanding ranges incorrectly, potentially leading to incorrect results or errors.

Key Assertions:

- The input string should be in the format 'start-end' (e.g. '1-3')
- The start value should be a digit and the end value should also be a digit
- All values between start and end should be integers
- If the range is invalid, an error message should be returned instead of expanding it
- The expanded list should contain all values in the original 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	10 lines (ranges: 81, 84-91, 95)

PASSED tests/test_ranges.py::TestExpandRanges::test_roundtrip 1ms 3

AI ASSESSMENT

Scenario: Verify that `compress_ranges` and `expand_ranges` functions return the same input when called in reverse order.

Why Needed: This test prevents a potential bug where the `roundtrip` functionality of these two functions is not working correctly, potentially leading to incorrect results or unexpected behavior.

Key Assertions:

- The output of `compress_ranges(original)` should be equal to `expand_ranges(compressed)`.
- The order of elements in `compressed` should match the original input list.
- All elements in `expanded` should be present in `original` and vice versa.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	27 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67, 81, 84-91, 93, 95)

PASSED tests/test_ranges.py::TestExpandRanges::test_single_number 1ms 3

AI ASSESSMENT

Scenario: The 'expand_ranges' function is expected to handle a single input ('5') and return a list containing only that number.

Why Needed: This test prevents a potential bug where the function does not correctly handle single numbers, potentially leading to incorrect results or errors.

Key Assertions:

- assert expand_ranges('5') == [5]
- assert expand_ranges('-5') == [-5]
- assert expand_ranges('abc') == []
- assert expand_ranges('5.0') == [5.0]

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	7 lines (ranges: 81, 84-87, 93, 95)

PASSED `tests/test_render.py::TestFormatDuration::test_milliseconds` 1ms 3

AI ASSESSMENT

Scenario: Test that the `format_duration` function correctly formats durations in milliseconds for values less than 1 second.

Why Needed: This test prevents a regression where the function does not format durations as expected for values less than 1 second.

Key Assertions:

- The function should return '500ms' when given a duration of exactly 0.5 seconds.
- The function should return '1ms' when given a duration of exactly 0.001 seconds.
- The function should return '0ms' when given a duration of exactly 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: Test the `format_duration` function with seconds input.

Why Needed: Prevents a potential bug where the function does not correctly format durations greater than or equal to 1 second.

Key Assertions:

- {'message': 'Input should be in decimal form (e.g., 1.23, 60.0)', 'description': 'The input duration is expected to be in decimal form.'}
- {'message': "Output should include 's' suffix for seconds", 'description': "The output of the function should include the 's' suffix for durations greater than or equal to 1 second."}
- {'message': 'Input values should not exceed 2 decimal places', 'description': 'The input duration values (1.23, 60.0) are expected to be rounded to two decimal places.'}
- {'message': 'Output values should match the expected format for seconds', 'description': 'The output of the function (1.23s, 60.00s) should match the expected format for durations greater than or equal to 1 second.'}

COVERAGE

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

Why Needed: This test prevents regression of CSS class mapping when an outcome is 'xfailed'.

Key Assertions:

- outcome-to-css-class mapping should be correct for all outcomes.
- outcome-to-css-class mapping should not include 'error' outcome.
- outcome-to-css-class mapping should map 'passed' and 'xpassed' to the correct CSS classes.
- outcome-to-css-class mapping should map 'failed' and 'xfailed' to the correct CSS classes.
- outcome-to-css-class mapping should map 'skipped' to the correct CSS class.
- outcome-to-css-class mapping should not include any other outcomes.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/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: The 'outcome_to_css_class' function is expected to handle unknown outcomes by returning a default CSS class.

Why Needed: This test prevents the unexpected behavior where an unknown outcome does not get a default CSS class.

Key Assertions:

- assert outcome_to_css_class('unknown') == 'outcome-unknown'
- assert outcome_to_css_class(None) == 'outcome-unknown'
- assert outcome_to_css_class(123) == 'outcome-unknown'
- assert outcome_to_css_class([]) == 'outcome-unknown'
- assert outcome_to_css_class({'a': 1, 'b': 2}) == 'outcome-unknown'
- assert outcome_to_css_class([1, 2]) == 'outcome-unknown'
- assert outcome_to_css_class(['a', 'b']) == 'outcome-unknown'
- assert outcome_to_css_class([]) == 'outcome-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/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 complete HTML document is rendered for the basic report scenario.

Why Needed: This test prevents rendering of an incomplete HTML document, which would cause the test to fail.

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' in the rendered HTML.
- The presence of '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: Test renders coverage for fallback HTML rendering.

Why Needed: Prevents a potential regression where the test fails to report coverage information when using fallback HTML rendering.

Key Assertions:

- The `report` object contains a `Summary` with total and passed counts.
- The `test::foo` nodeid is included in the rendered HTML.
- 5 lines are reported as being rendered in the 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: The test verifies that the LLM annotation is included in the rendered HTML report.

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

Key Assertions:

- The string "Tests login flow" should be present in the rendered HTML report.
- The string "Prevents auth bypass" should also be present in the rendered HTML report.
- The LLM annotation should be included in the rendered HTML report, indicating that it is properly configured and functioning as expected.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	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: Verifies that the source coverage summary is included in the rendered HTML.

Why Needed: This test prevents regression where the source coverage summary was not displayed correctly.

Key Assertions:

- The 'Source Coverage' section should be present in the rendered HTML.
- The text 'src/foo.py' should be included in the 'Source Coverage' section.
- The percentage of covered code ('80.0%') should be displayed 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	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: Test renders summary with xfailed and xpassed counts.

Why Needed: This test prevents rendering of the summary if there are any failed tests (xfailed) or passed tests (xpassed).

Key Assertions:

- The 'XFailed' tag is present in the rendered HTML.
- The 'XPassed' tag is also present in the rendered HTML.
- Both 'XFailed' and 'XPassed' tags are included 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	50 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-127, 131-132, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED tests/test_report_writer.py::TestComputeSha256::test_different_content 1ms 3

AI ASSESSMENT

Scenario: Test Compute Sha256 with different content

Why Needed: Prevents a potential bug where the same input produces the same hash.

Key Assertions:

- The function `compute_sha256` should produce different hashes for two different inputs.
- The function `compute_sha256` should not produce the same hash for the same input.
- The output of `compute_sha256(b'hello')` should be different from `compute_sha256(b'world')`.
- The output of `compute_sha256(b'hello') != compute_sha256(b'world')` should be True.
- The function should raise an exception if the inputs are the same (e.g., `compute_sha256(b'hello') == compute_sha256(b'hello')`)
- The function should not produce a hash for the same input multiple times (e.g., `compute_sha256(b'hello') != compute_sha256(b'hello')`)

COVERAGE

src/pytest_llm_report/collector.py	14 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 input produces consistent hash and correct length.

Why Needed: This test prevents a potential issue where the hash function may produce incorrect results or be slow for very small inputs.

Key Assertions:

- The two computed hashes are equal (i.e., they have the same value).
- Both hashes should have a length of 64 bytes (the expected SHA256 hex length).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

PASSED tests/test_report_writer.py::TestReportWriter::test_build_run_meta 5ms 4

AI ASSESSMENT

Scenario: Test that the build_run_meta method includes version info for the test case.

Why Needed: This test prevents regression where the pytest version is not included in the run metadata.

Key Assertions:

- The duration of the test should be 60 seconds.
- The pytest version should have a non-zero value.
- The plugin version should be '0.1.0'.
- The Python version should also be included.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/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 Summary class correctly counts all outcome types and their corresponding counts.

Why Needed: This test prevents a regression where the Summary class incorrectly reports an outcome type as 'xfailed' when it actually means 'xpassed'.

Key Assertions:

- The total count of outcomes should be 6 (1 passed, 1 failed, 1 skipped, 1 xfailed, 1 xpassed, 1 error).
- The number of passed outcomes should be 1.
- The number of failed outcomes should be 1.
- The number of skipped outcomes should be 1.
- The number of xfailed outcomes should be 1.
- The number of xpassed outcomes should be 1.
- The number of error outcomes should be 1.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	19 lines (ranges: 156-158, 312, 314-315, 317-328, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_build_summary_counts` 1ms 4

AI ASSESSMENT

Scenario: The test verifies that the `test_build_summary_counts` function correctly counts outcomes in a report.

Why Needed: This test prevents regression where the count of passed and failed tests is incorrect due to missing or incorrectly counted skipped tests.

Key Assertions:

- total should be equal to 4 (2 passed, 1 failed, 1 skipped)
- passed should be equal to 2
- failed should be equal to 1
- skipped should be equal to 1

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 a new ReportWriter instance is created with the provided configuration.

Why Needed: This test prevents a potential bug where a new ReportWriter instance is not initialized correctly, leading to incorrect warnings and artifacts being tracked.

Key Assertions:

- The `config` attribute of the `ReportWriter` instance should be equal 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.

COVERAGE

src/pytest_llm_report/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 that ReportWriter writes a report with all tests.

Why Needed: This test prevents regression where the report does not include all tests, potentially causing confusion or missing important information.

Key Assertions:

- The length of the report.tests list should be equal to 2.
- The total number of tests in the summary 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 the `test_write_report_includes_coverage_percent` function to verify that it includes total coverage percentage in the generated report.

Why Needed: This test prevents a potential bug where the report does not include the total coverage percentage, potentially misleading users about the completeness of the report.

Key Assertions:

- The `summary.coverage_total_percent` attribute of the report object should be equal to the provided `coverage_percent` value.
- The `report` object's `summary` attribute should have a `coverage_total_percent` attribute that matches the given value.
- The `report.write_report()` method should generate an output with a `summary` section containing the correct coverage percentage.
- The `report.summary.coverage_total_percent` property should be equal to the provided value after writing the report.
- The test should fail if the generated report does not include the total coverage percentage.
- The test should pass if the generated report includes the total coverage percentage correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	93 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-199, 202-206, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_includes_source_coverage` 5ms 4

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_includes_source_coverage verifies that the report includes 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` should be 1.
- The file path of `report.source_coverage[0]` should match `source_coverage[0].file_path`.
- All statements in `source_coverage` should be covered by the report.
- At least one statement out of `missed` and `covered` should be missed by the report.
- The coverage percentage should be greater than or equal to 87.5%
- Covered ranges should match `source_coverage[0].covered_ranges` exactly.
- Missed ranges should match `source_coverage[0].missed_ranges` exactly.
- All statements in `source_coverage` should have a corresponding range in the report (e.g., `1-4`, `6-7`) if applicable.

COVERAGE

src/pytest_llm_report/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 test writes a merged coverage report.

Why Needed: This test prevents regression where the coverage is not properly merged into tests.

Key Assertions:

- The number of lines covered by each test should be exactly 1.
- Each file path in the coverage data should match the file path of the first test.
- All line ranges and counts should be present in the coverage data for the first 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	94 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186-189, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_atomic_write_fallback` 7ms 5

AI ASSESSMENT

Scenario: Test the fallback to direct write when atomic write fails.

Why Needed: To prevent a regression where the test relies on an atomic write operation failing, which would cause the report to be written directly instead of falling back to a file.

Key Assertions:

- The `report.json` file exists at the expected location.
- At least one warning message has the code 'W203'.
- The `writer.warnings` list contains warnings with code 'W203'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	67 lines (ranges: 229-231, 233, 235, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	125 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506-507, 509-512, 515-516)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_creates_directory_if_missing` 7ms 5

AI ASSESSMENT

Scenario: Test should create output directory if it doesn't exist.

Why Needed: Prevents a potential issue where the report writer fails to create an output directory if it does not exist.

Key Assertions:

- The test checks that `tmp_path / 'subdir' / 'report.json'` exists after writing the report.
- The test checks that `tmp_path / 'subdir' / 'report.json'` is a file.
- The test checks that `tmp_path / 'subdir' / 'report.json'` has been created with the correct permissions.
- The test checks that the output directory exists even if there are no tests in it.
- The test checks that the report writer creates the output directory before writing the report.
- The test checks that the report writer does not create an empty directory.
- The test checks that the report writer raises an exception when trying to write a non-existent 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/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 the `test_ensure_dir_failure` function to ensure that it captures warnings when directory creation fails.

Why Needed: This test prevents a potential issue where the `ReportWriter` class does not handle directory creation failures correctly and may silently ignore or mask these errors.

Key Assertions:

- The `writer.warnings` list should contain at least one warning with code 'W201' when the directory is created.
- The `writer.warnings` list should contain a single warning with code 'W201' when the directory creation attempt fails.
- Any warnings raised by the `writer.warnings` list should have a code of 'W201'.
- The `writer.warnings` list should not be empty after attempting to create the directory.
- The `writer.warnings` list should contain at least one warning with an error message that indicates the directory creation 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)
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: Test 'test_git_info_failure' verifies that the `get_git_info` function handles git command failures gracefully by returning `None` for both SHA and dirty flags.

Why Needed: This test prevents a regression where the `get_git_info` function fails to return expected values when encountering git command failures.

Key Assertions:

- The `get_git_info` function should not raise an exception if it encounters a git command failure.
- The `sha` variable should be set to `None` after calling `get_git_info()`.
- The `dirty` variable should be set to `None` after calling `get_git_info()`.
- The `get_git_info()` function should not raise an exception when encountering a git command failure using the `subprocess.check_output` mock.
- The `sha` and `dirty` variables should still have their expected values after the test.
- The `get_git_info()` function should return `None` for both SHA and dirty flags 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)
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` 31ms 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 at the specified path.
- The 'report.html' file should contain expected content as described in the test.
- All expected text should be present in the 'report.html' file.
- The report writer should be able to write a report with tests that fail or are skipped.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/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` 33ms 5

AI ASSESSMENT

Scenario: The test verifies that the report includes xfail outcomes in its HTML summary.

Why Needed: This test prevents regression by ensuring that xfail outcomes are included in the HTML summary, which can help with debugging and reporting issues.

Key Assertions:

- The 'XFAILED' keyword is present in the HTML summary.
- The 'XFailed' keyword is present in the HTML summary.
- The 'XPASSED' keyword is present in the HTML summary.
- The 'XPassed' keyword is present in the HTML summary.

COVERAGE

src/pytest_llm_report/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 report data.

Why Needed: This test prevents regression in case the `report_json` attribute is not provided to the `Config` constructor, causing an error when trying to create a JSON file.

Key Assertions:

- The `report.json` file should be created at the specified path.
- At least one artifact should be tracked in the report.
- The length of the artifacts list should be greater than zero.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	78 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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` 35ms 5

AI ASSESSMENT

Scenario: Should create PDF file when Playwright is available.

Why Needed: Prevents a bug where the test fails to write the PDF file because the playwright module is not imported or is not available.

Key Assertions:

- The `report.pdf` attribute of the test case should be set to a valid path.
- Any artifacts created by the report writer should have the same path as the `report.pdf` attribute.
- All files in the `report.pdf` directory should exist.
- The `report.pdf` file should not be empty.
- The `report.pdf` file should contain at least one artifact.
- The `report.pdf` file should not be a symbolic link.
- Any artifacts created by the report writer should have a valid path and not be a directory.
- The `report.pdf` file should not be deleted or modified after writing.

COVERAGE

src/pytest_llm_report/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 verifies that a warning is raised when Playwright is missing for PDF output.

Why Needed: This test prevents the 'Warning: Could not find module 'playwright'' error, which would prevent the report from being written to PDF.

Key Assertions:

- The file 'report.pdf' should exist after the test.
- Any warning with code WarningCode.W204_PDF_PLAYWRIGHT_MISSING should be present in the list of warnings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	98 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230-231, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 401-405, 408)

PASSED `tests/test_report_writer_coverage_v2.py::test_report_writer_ensure_dir_creation` 1ms 4

AI ASSESSMENT

Scenario: Test ensures directory creation when report HTML file exists.

Why Needed: This test prevents a potential issue where the report writer does not create the required directory if it already exists, potentially leading to unexpected behavior or errors.

Key Assertions:

- The `tmp_dir` should exist before attempting to write the report.
- At least one warning code ('W202') should be present in the report.
- The `tmp_dir` should not be deleted after writing the report.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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` 11ms 5

AI ASSESSMENT

Scenario: Test verifies that report_writer_metadata_skips function prevents skipping of metadata when reports are disabled.

Why Needed: This test prevents a bug where the report writer skips metadata, potentially leading to incorrect or incomplete report data.

Key Assertions:

- The 'start_time' key should be present in the metadata.
- The 'llm_model' key should not be present in the metadata when reports are disabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	36 lines (ranges: 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	67 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300)

PASSED tests/test_schemas.py::TestAnnotationSchema::test_from_dict_full 1ms 3

AI ASSESSMENT

Scenario: Test that `AnnotationSchema.from_dict` can create a valid annotation from a dictionary with all required fields.

Why Needed: Prevents regression in case of missing or invalid data.

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: Test that the `to_dict` method returns all required fields for a schema.

Why Needed: Prevents regression in cases where authentication is not properly handled by the API.

Key Assertions:

- assert data['scenario'] == 'Verify login',
- assert data['why_needed'] == 'Catch auth bugs',
- assert data['key_assertions'] == ['assert 200', 'assert 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/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: Verify that an HTML report is generated for the test `simple()`.

Why Needed: Prevents a potential regression where the test does not generate an HTML report.

Key Assertions:

- The file path to the generated report should exist.
- The content of the report should contain '' and 'test_simple' keywords.
- The keyword 'test_simple' should be present in the report's content.

COVERAGE

src/pytest_llm_report/collector.py	39 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)

src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	101 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_summary_counts_all_statuses` 121ms 7

AI ASSESSMENT

Scenario: test_html_summary_counts_all_statuses verifies that the HTML summary counts include all statuses.

Why Needed: This test prevents regression where the report does not include all statuses, potentially leading to confusion or incorrect reporting.

Key Assertions:

- The 'Total Tests' label should be present in the HTML summary.
- Each status (Passed, Failed, Skipped) should have a corresponding count in the HTML summary.
- XFailed and XPassed should also be included in the HTML summary for expected failures and successes.
- Errors should only include one entry with the 'Error' label.
- The 'Total Tests' count should match the total number of tests run.
- Each status's count should match its corresponding label (e.g. 'Passed' has 1 count, not 2).
- If a test is skipped, it should be marked as such in the HTML summary.

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` 74ms 7

AI ASSESSMENT

Scenario: The JSON report is created and its existence, schema version, summary statistics are verified.

Why Needed: This test prevents a potential bug where the Pytest report generator does not create a JSON file for reports.

Key Assertions:

- The `report_path.exists()` assertion checks if the generated report exists at the specified path.
- The `data` variable is loaded from the report file using `json.loads()`.
- The `schema_version` key in the `data` dictionary is checked to ensure it matches the expected value.
- The `summary` dictionary contains the correct number of 'passed' and 'failed' items.
- The total count of test cases is verified by checking the 'total' key in the summary dictionary.
- The passed count is verified by checking the 'passed' key in the summary dictionary.
- The failed count is verified by checking the 'failed' key in the summary dictionary.

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` 82ms 13

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report.

Why Needed: Prevents regressions and ensures accurate test results by including LLM annotations in the report.

Key Assertions:

- The 'scenario' key is present in the report data.
- The 'why_needed' key is present in the report data.
- The 'llm_annotation' key contains the expected value for the 'scenario' key.
- The 'choices' list within the 'llm_annotation' dictionary contains the expected values for the 'message' and 'content' keys.
- The 'key_assertions' list within the test data is present in the report data.
- The 'tests[0]' dictionary within the report data contains the expected value for the 'scenario' key.
- The 'llm_annotation' dictionary within the test data has a valid JSON representation.

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: This test prevents regression where LLM errors are not reported correctly.

Key Assertions:

- The report contains the string 'LLM error'.
- The report contains the string 'boom'.

COVERAGE

src/pytest_llm_report/cache.py	12 lines (ranges: 39-41, 53, 55-56, 86, 88, 118-119, 121, 153)
src/pytest_llm_report/collector.py	39 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/llm/annotator.py	73 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137-139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198-201, 203)
src/pytest_llm_report/llm/base.py	21 lines (ranges: 52-53, 72, 75, 80, 107, 110-111, 128, 136, 147, 165, 167, 175, 245, 247, 249, 252, 257-258, 260)
src/pytest_llm_report/llm/litellm_provider.py	25 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 78-79, 81-82, 84-85, 94-95, 97)
src/pytest_llm_report/options.py	47 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)

src/pytest_llm_report/plugin.py	186 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 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 60ms 7

AI ASSESSMENT

Scenario: Test the LLM opt-out marker functionality.

Why Needed: Prevents a regression where the LLM opt-out marker does not record the test.

Key Assertions:

- The 'llm_opt_out' marker should be applied to the test function.
- The test function should have an attribute 'llm_opt_out' set to True.
- The report generated by pytester should contain a single test with 'llm_opt_out' as False.
- The number of tests in the report should be 1.
- The first test in the report should have 'llm_opt_out' set to True.
- The 'llm_opt_out' attribute should be present on the test function.
- The 'llm_opt_out' attribute should be a boolean value (True or False).

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 58ms 7

AI ASSESSMENT

Scenario: Test the requirement marker functionality.

Why Needed: This test prevents a potential bug where the requirement marker is not recorded correctly, potentially leading to incorrect reporting of tests.

Key Assertions:

- The `pytest.mark.requirement` decorator should be applied to the test function with two required requirements.
- The `pytester.makepyfile()` method should create a new file with the correct syntax for marking requirements.
- The `report.json` path should be created correctly and contain one test result with both requirement markers.
- The JSON data should contain a single test with both requirement markers.
- The 'requirements' key in the test data should contain both required requirements.
- The 'REQ-001' and 'REQ-002' strings should be present in the 'requirements' list of the test result.

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` 66ms 7

AI ASSESSMENT

Scenario: Test that multiple xfail tests are recorded in the report.

Why Needed: Prevents regression by ensuring that all xfailed tests are captured in the report.

Key Assertions:

- The number of xfailed tests is equal to 2 (test_xfail_one and test_xfail_two).
- All xfailed tests have an outcome of '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 58ms 7

AI ASSESSMENT

Scenario: Test that skipping tests prevents incorrect reporting of skipped outcomes.

Why Needed: This test prevents regression where the number of skipped tests is reported incorrectly in the report.

Key Assertions:

- The 'skipped' key in the report.json file should contain a count equal to the number of tests marked as skip.
- The value of the 'skipped' key should be an integer (1 or 0).
- If no tests are skipped, the 'skipped' key should be empty.
- If all tests pass, the 'skipped' key should be 0.
- If a test is marked as skip but not run, the 'skipped' key should still contain a count of 1.
- The report.json file should contain a valid JSON structure with the correct keys and values.

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` 62ms 7

AI ASSESSMENT

Scenario: Verifies that xfailed tests are recorded in the report.

Why Needed: Prevents regression where only passed tests are reported.

Key Assertions:

- The 'summary' key under 'xfailed' should be present and have a value of 1.
- The 'xfailed' count should match the actual number of xfailed tests in the report.
- The test function 'test_xfail()' should not raise an AssertionError.

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` 63ms 7

AI ASSESSMENT

Scenario: Test parameterized tests are recorded separately.

Why Needed: This test prevents regression by ensuring that all parametrized tests are run and reported correctly.

Key Assertions:

- The total number of tests passed should be 3 (1 > 0, 2 > 0, 3 > 0).
- All parametrized tests should have been recorded in the report.json file.
- The summary data should contain 'total' and 'passed' keys with values equal to 3.

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` 52ms 3

AI ASSESSMENT

Scenario: The test verifies that the CLI help text includes usage examples.

Why Needed: This test prevents a potential bug where the help message does not include any usage examples, making it difficult for users to understand how to use the plugin.

Key Assertions:

- The output of `pytester.runpytest('--help')` should contain lines that match the pattern `*Example:*--llm-report*`.
- The output of `pytester.runpytest('--help')` should not contain any lines that do not match this pattern.
- The text 'Example:' should be present in the help message.
- The text '--llm-report' should be present in the help message.
- The text '*llm-report*' should be present in the help message.
- The output of `pytester.runpytest('--help')` should not contain any lines that are not part of the expected pattern.
- The output of `pytester.runpytest('--help')` should have a total length greater than 0 characters.
- The output of `pytester.runpytest('--help')` should have a total length less than or equal to 100 characters.
- The output of `pytester.runpytest('--help')` should not contain any whitespace characters.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 46ms 3

AI ASSESSMENT

Scenario: Verify that LLM markers are registered and their presence is checked during the pytest run.

Why Needed: Prevents a potential bug where LLM markers are not properly registered, potentially leading to test failures or incorrect results.

Key Assertions:

- The 'markers' marker should be present in the output of `pytester.runpytest`.
- The presence of 'llm_opt_out', 'llm_context', and 'requirement' markers should be checked in the output.
- The markers should be correctly matched against their expected strings ('*llm_opt_out*', '*llm_context*', and '*requirement*').

COVERAGE

src/pytest_llm_report/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` 53ms 3

AI ASSESSMENT

Scenario: Verify that the plugin is successfully registered via pytest11 and its help output matches the expected pattern.

Why Needed: This test prevents a potential issue where the plugin's registration fails or produces unexpected results due to incorrect usage of pytest11.

Key Assertions:

- The `pytester.runpytest` method should call `--help` with the expected output.
- The `stdout.fnmatch_lines` method should match the expected pattern for the help output.
- The `result.stdout` attribute should contain a non-empty string after running pytest11 with the `--help` option.
- The `result.stdout` attribute should not raise any exceptions during execution.
- The `result.stdout` attribute should contain the correct number of lines matching the expected pattern.
- The `stdout.fnmatch_lines` method should match the expected pattern for the help output.
- The `pytester.runpytest` method should call `--help` with the correct options and arguments.
- The `pytester.runpytest` method 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/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` 87ms 7

AI ASSESSMENT

Scenario: Test that special characters in nodeid are handled correctly by Pytest.

Why Needed: This test prevents a potential crash and ensures the HTML file is valid.

Key Assertions:

- The string '' should be present in the report.html file.
- The string 'html' should be present in the report.html file.
- The report.html file should exist.
- The content of the report.html file should contain the string '
- The content of the report.html file should not contain any special characters in the nodeid.

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 boundary of exactly one minute.

Why Needed: Prevents regression in formatting durations to exactly one minute.

Key Assertions:

- The result of `format_duration(60.0)` should be '1m 00s' (or '1:00s' for human-readable format).
- The function does not return an error or raise an exception when the input is less than one minute.
- The function correctly handles cases where the input is exactly one minute, such as `format_duration(60.0)`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

PASSED `tests/test_time.py::TestFormatDuration::test_microseconds_format` 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 500 microseconds.

Why Needed: Prevents regression in formatting sub-millisecond durations as microseconds.

Key Assertions:

- The output should contain 'µs' to indicate microsecond format.
- The actual output should be equal to '500µs'.
- The function correctly formats the duration of 0.0005 seconds as 500 microseconds.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 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 regression when the duration is less than one second, where the format_duration function should return '0s' instead of an incorrect value.

Key Assertions:

- The result string contains the word 'ms' (milliseconds).
- The result equals '500.0ms'.

COVERAGE

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

Why Needed: This test prevents regression where durations are not correctly formatted as 'mm:ss'.

Key Assertions:

- The function `format_duration` returns a string in the format 'mm:ss' for durations over one minute.
- The function `format_duration` correctly handles cases where the duration is greater than or equal to one minute.
- The function `format_duration` preserves the original seconds value of the input duration.
- The function `format_duration` does not truncate the minutes part if it's less than 1.
- The function `format_duration` returns a string in the format 'mm:ss' even for durations with fractional parts (e.g., 90.5s).
- The function `format_duration` handles cases where the input duration is zero or negative 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/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 multiple minutes.

Why Needed: Prevents a potential bug where the function incorrectly formats minutes as seconds.

Key Assertions:

- The function should correctly format multiple minutes into 'mm:ss' format.
- The function should handle cases where the input is not an integer.
- The function should preserve trailing zeros in the output.
- The function should correctly handle decimal numbers (e.g., 3.5).
- The function should support negative inputs (e.g., -185.0).
- The function should raise a TypeError for non-numeric input values.
- The function should return an error message when the input is 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/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 regression in the `format_duration` function when given a duration that is less than or equal to zero.

Key Assertions:

- The output of `format_duration(1.0)` should be '1.00s'.
- The duration string should not contain any decimal places.
- The unit should be seconds (s).
- The function should raise a ValueError for negative durations.
- The function should return an empty string for zero duration.
- The function should handle non-numeric inputs correctly.
- The function should handle very large or very small numbers correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED tests/test_time.py::TestFormatDuration::test_seconds_format 1ms 3

AI ASSESSMENT

Scenario: Test the 'seconds' format for values between 0 and 59.

Why Needed: This test prevents regression when seconds are formatted to two decimal places.

Key Assertions:

- The function should return a string with 's' appended to the number of seconds.
- The function should return the value as a string in the format 'X.Xs'.
- The function should handle values between 0 and 59 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/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED tests/test_time.py::TestFormatDuration::test_small_milliseconds 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 1 millisecond.

Why Needed: This test prevents a potential bug where the function returns incorrect results for small durations due to precision issues.

Key Assertions:

- The output should be '1.0ms' when the input is 1 millisecond.
- The function should return an integer value (1 in this case) because it's a duration.

COVERAGE

src/pytest_llm_report/collector.py	14 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: Tests the `format_duration` function with a duration of 1 microsecond.

Why Needed: This test prevents a potential bug where the function returns incorrect results for very small durations.

Key Assertions:

- The result of `format_duration(0.000001)` should be '1µs'.
- The unit of the result should be microseconds (µs).
- The function should handle negative values correctly.
- The function should return an empty string for zero duration.
- The function should raise a ValueError for invalid input (e.g., non-numeric value).
- The function should preserve the original sign of the input value.
- The function should support floating-point arithmetic accurately.

COVERAGE

src/pytest_llm_report/collector.py	14 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 `iso_format` function 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)` is equal to '2024-01-15T10:30:45+00:00'.
- The `tzinfo` attribute of the datetime object `dt` is set to `UTC`.
- The resulting string does not contain any timezone offset information.
- The resulting string represents a valid ISO 8601 date and time format.
- The function correctly handles edge cases where the datetime object has a timezone that is not UTC.
- The function raises an error when given a datetime object with a non-UTC timezone.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

PASSED tests/test_time.py::TestIsoFormat::test_formats_naive_datetime 1ms 3

AI ASSESSMENT

Scenario: Test the naive datetime format by verifying it matches a standard ISO format.

Why Needed: Prevents regression in date formatting when no timezone is specified.

Key Assertions:

- The result of `iso_format(dt)` should be equal to '2024-06-20T14:00:00'.
- The time part of the result should match exactly with '14:00:00'.
- The date part of the result should match exactly with '2024-06-20'.
- The timezone offset should be None (no timezone specified).

COVERAGE

src/pytest_llm_report/collector.py	14 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 datetime objects containing microseconds.

Why Needed: Prevents a potential issue where microseconds are not correctly formatted as ISO 8601.

Key Assertions:

- The output of `iso_format(dt)` should contain '123456' in its string representation.
- The value '123456' should be present in the output string.
- The format string used by `iso_format` should include microseconds (e.g., '%Y-%m-%dT%H:%M:%S.%f%z').
- The function should correctly handle datetime objects with microseconds (e.g., 2024-03-10T08:15:30.123456+00:00).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

PASSED tests/test_time.py::TestUtcNow::test_has_utc_timezone 1ms 3

AI ASSESSMENT

Scenario: Verifies that the `utc_now()` function returns a datetime object with an associated UTC timezone.

Why Needed: Prevents regression in tests where the `datetime` module's default timezone is not set to UTC.

Key Assertions:

- result.tzinfo is not None
- result.tzinfo == UTC
- assert result.tzinfo is None would 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/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 a potential issue where the `utc_now()` function returns an incorrect or outdated time due to a timing-related bug.

Key Assertions:

- The returned time should be within a certain range (e.g., 1 second) of the current UTC time.
- The returned time should not exceed the current UTC time by more than 1 second.
- The returned time should not be less than the current UTC time by more than 1 second.
- The `utc_now()` function should return a time that is within the same timezone as the test environment.
- The `utc_now()` function should not introduce any significant timing-related errors or inconsistencies.
- The `utc_now()` function should handle edge cases such as midnight, noon, and midnight of the next day 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/time.py	1 lines (ranges: 15)

PASSED tests/test_time.py::TestUtcNow::test_returns_datetime 1ms 3

AI ASSESSMENT

Scenario: The `utc_now()` function returns a datetime object.

Why Needed: This test prevents a potential bug where the `utc_now()` function does not return a datetime object when called without any arguments.

Key Assertions:

- result is an instance of datetime
- result is a valid datetime object
- result is not None
- result has a valid timezone
- result has a valid date and time
- result has a valid time
- result has a valid year

COVERAGE

src/pytest_llm_report/collector.py	14 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	-