Test Report

Run ID: 20966792397-py3.12 • Generated: 2026-01-13 17:49:04 • Duration: 34.52s
Plugin: v0.1.0 (2f498263985a34902252c53c11fb820445bd8f21) [dirty]
Repo: v0.1.0 (3cc50ec73009d5fecab87cec0ec9b31f805a0863)
LLM: ollama / llama3.2:1b (minimal context, 380 annotated, 6 errors)

92.91%
Total Coverage

387	387	0	0
TOTAL TESTS	PASSED	FAILED	SKIPPED

0	0	0
XFAILED	XPASSED	ERRORS

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

AI ASSESSMENT

Scenario: Test the aggregation of multiple policy reports to ensure both tests are included in the aggregate report.

Why Needed: This test prevents regression where only one or no policy reports are aggregated, potentially leading to incomplete or missing test results.

Key Assertions:

- The `retain` parameter is set to 'all' and all retained tests from both policy reports are present in the aggregate result.
- The number of tests in the aggregate result matches the expected value (2) even when one or both policy reports fail.
- All retained test cases from both policy reports are included in the `tests` list within the aggregate result.
- No empty lists are present in the `tests` list within the aggregate result.
- The `retain` parameter is correctly applied to both policy reports and their aggregated results.
- The aggregation process does not introduce any new test cases or remove existing ones without a valid reason.
- The expected number of tests in the aggregate result matches the actual count when one or both policy reports fail.

COVERAGE

src/pytest_llm_report/aggregation.py	69 lines (ranges: 52, 55-56, 59, 61-63, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 130-131, 134, 141, 146, 148-153, 155, 157-159, 170, 217, 219-223, 235, 245, 248-249, 251, 253, 275-278, 280)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: Prevents a bug where the aggregate function throws an exception or raises an error when a non-existent directory is passed.

Key Assertions:

- The `aggregator.aggregate()` method should return `None` when the specified directory does not exist.
- The `pathlib.Path.exists()` mock object should return `False` for the specified directory.
- No exception or error should be raised by the `aggregator.aggregate()` method.
- The aggregate function should not throw an exception or raise an error even if the directory exists but is empty.
- The aggregate function should not throw an exception or raise an error when the directory does not exist but has a valid path.
- The aggregate function should handle non-existent directories correctly and return `None` without raising any exceptions.

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 correctly picks the latest policy for a given test case.

Why Needed: This test prevents regression where the previous implementation would incorrectly pick an older policy for a new test case.

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 an `is_aggregated` attribute set to True.
- The `run_meta.run_count` attribute should be equal to 2.
- The `summary.passed` attribute should be equal to 1 (indicating one test passed).
- The `summary.failed` attribute should be equal to 0 (indicating no tests failed).

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 return an empty list when no directory configuration is provided.

Why Needed: This test prevents a potential bug where the aggregator function returns an error or raises an exception without providing any aggregation results.

Key Assertions:

- agg.aggregate() should be called with no arguments.
- agg.aggregate() should return None.
- agg.aggregate() should not raise any exceptions when aggregate_dir is None.
- agg.aggregate() should handle the case where aggregate_dir is an empty string or None without raising an exception.
- agg.aggregate() should use the provided mock_config object to configure the aggregator instance.
- agg.aggregate() should not use any default aggregation strategy if aggregate_dir is None.
- agg.aggregate() should return a list of aggregated values with no elements when aggregate_dir is None.

COVERAGE

src/pytest_llm_report/aggregation.py	3 lines (ranges: 44, 52-53)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The `aggregate` function should not be called when there are no reports.

Why Needed: This test prevents a potential bug where the `aggregate` function is called with an empty list of reports, potentially causing unexpected behavior or errors.

Key Assertions:

- aggregator.aggregate() should return None when there are no reports.
- the returned value should be None.
- no exception should be raised when calling `aggregate()`
- the function call should not modify the original list of reports.
- the function call should not raise an error if the input is invalid.
- the function call should return None without raising any exceptions.
- the returned value should be a single value (None) rather than a list.
- the `aggregate` function should not modify the original list of reports.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- Coverage was properly deserialized with correct file paths and line counts.
- LLM annotation was properly deserialized with correct scenario, why needed, key assertions, and confidence.
- Test can be re-serialized without issues (previously failed due to regression)
- Serialized report contains required coverage and LLM annotations.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the aggregation function with source coverage verification.

Why Needed: This test prevents regression that may occur when the source coverage is not properly aggregated.

Key Assertions:

- The source coverage summary should be deserialized correctly from a JSON report.
- The correct file path of the source code should be matched in the aggregated result.
- A SourceCoverageEntry object should be created with the expected attributes (file_path, statements, missed, covered, coverage_percent, covered_ranges, missed_ranges).
- The length of the source_coverage list should be 1 as per the test scenario.
- The isinstance function should correctly identify a SourceCoverageEntry object in the result.
- The file path attribute of the first SourceCoverageEntry object should match the expected value (src/foo.py).

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: This test prevents a bug where the aggregator fails to load coverage data when the LLM's default source file is not provided.

Key Assertions:

- Verify that _load_coverage_from_source() returns None when llm_coverage_source is set to None.
- Verify that _load_coverage_from_source() raises a UserWarning when llm_coverage_source does not exist.
- Verify that _load_coverage_from_source() successfully loads coverage data from a mock .coverage file when llm_coverage_source is set to .coverage.
- Verify that the mocked CoverageMapper returns the correct entry for SourceCoverageEntry.
- Verify that the mocked CoverageMapper calls the load method on the mocked Coverage object.
- Verify that the mocked CoverageMapper calls the map_source_coverage method on the mocked Coverage object with the mock entry.
- Verify that _load_coverage_from_source() returns a result with exactly one entry and 80.0 coverage percentage when llm_coverage_source is set to .coverage.
- Verify that the returned entries are of type SourceCoverageEntry.

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: Recalculate Summary

Why Needed: Prevents regression in aggregation logic when multiple tests are passed or failed simultaneously.

Key Assertions:

- The total number of test results should be equal to the total number of nodes tested.
- The passed count should match the expected number of passed test results.
- The failed count should match the expected number of failed test results.
- The skipped count should match the expected number of skipped test results.
- The xfailed count should match the expected number of xfailed test results.
- The xpassed count should match the expected number of xpassed test results.
- The error count should be 1 (as per the latest summary provided).
- The coverage percentage should remain preserved even when multiple tests are passed or failed simultaneously.

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 report prevents the test from counting it as a valid aggregation run.

Why Needed: This test ensures that the `aggregate` function correctly handles reports with missing fields and invalid JSON.

Key Assertions:

- The `aggregate` function should not count the skipped report as a valid aggregation run.
- The `aggregate` function should raise a warning when it encounters an invalid JSON report file.
- The test should fail to count the skipped report if its fields are missing.
- The test should only count the valid report in the aggregate result.
- The `aggregate` function should not include the invalid JSON report in the aggregation output.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents regression in case of multiple tests failing, as it ensures that the aggregator calculates the correct total duration and coverage percentage.

Key Assertions:

- summary.total == 2 (correct number of tests)
- summary.passed == 1 (correct number of passed tests)
- summary.failed == 1 (correct number of failed tests)
- summary.coverage_total_percent == 88.5 (correct coverage percentage)
- summary.total_duration == 3.0 (correct total duration)

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verifies that cached tests are skipped when using the annotator.

Why Needed: This test prevents a regression where cached annotations would be re-run unnecessarily.

Key Assertions:

- The function `test_cached_tests_are_skipped` is called with mock providers, cache and assembler.
- The `mock_provider`, `mock_cache` and `mock_assembler` objects are created without being used in the test.
- The `test_cached_tests_are_skipped` method does not call any of its dependencies (mock providers, cache and assembler).
- The `test_cached_tests_are_skipped` method returns a mock object that is not an instance of the original function.
- The `test_cached_tests_are_skipped` method does not raise any exceptions during execution.
- The `test_cached_tests_are_skipped` method does not modify any external state.
- The `test_cached_tests_are_skipped` method does not call any other test methods.
- The `mock_provider`, `mock_cache` and `mock_assembler` objects are created with default values.

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: Testing concurrent annotation functionality.

Why Needed: Prevents potential performance regressions and ensures thread safety in concurrent scenarios.

Key Assertions:

- Ensures that the annotator does not annotate multiple documents simultaneously, preventing potential performance issues.
- Verifies that the cache is accessed sequentially by the annotator, ensuring thread safety.
- Checks for any exceptions raised during annotation operations to prevent unexpected behavior.
- Validates that the assembler is properly initialized before concurrent annotation attempts are made.
- Ensures that the annotator does not annotate a document while another annotator is still processing it in the cache.
- Verifies that the annotator's output is consistent across multiple threads, ensuring thread safety and correctness.
- Checks for any synchronization issues or bottlenecks that may arise from concurrent annotation operations.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test concurrent annotation handles failures to verify

Why Needed: This test prevents a regression where the annotator may fail to handle failures in the annotation process.

Key Assertions:

- The mock provider is not called with an error message when annotated fails.
- The mock cache is not modified when annotated fails.
- The mock assembler does not raise an exception when annotated fails.
- The capture fixture is not captured when annotated fails.
- The mock provider's return value is not None when annotated fails.
- The mock cache's get method returns None when annotated fails.
- The mock assembler's error message is not 'Error annotation failed'.
- The annotator does not raise an exception when annotated fails.

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 functionality.

Why Needed: This test prevents regression in the annotator's progress reporting feature when using mock providers and a cache.

Key Assertions:

- mock_provider is not None.
- mock_cache is not None.
- mock_assembler is not None.

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: Testing sequential annotation functionality in the annotator.

Why Needed: This test prevents regression when using sequential annotation with multiple providers and caches.

Key Assertions:

- Verify that the annotator correctly handles sequential annotation with multiple providers and caches.
- Check if the annotator properly updates the cache after sequential annotation.
- Ensure that the annotator does not throw an exception when encountering a provider or cache issue during sequential annotation.
- Verify that the annotator preserves the original order of annotations in the sequence.
- Test the case where multiple providers are used and the annotator correctly handles their sequential annotation.
- Check if the annotator throws an error when the number of providers exceeds the maximum allowed.
- Verify that the annotator does not throw an exception when encountering a provider or cache issue during sequential annotation with a large number of providers.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the annotator's functionality may be unexpectedly disabled without proper indication.

Key Assertions:

- The `config` object passed to `annotate_tests` has an 'enabled' attribute set to False.
- The `annotate_tests` function does not attempt to annotate any tests when the LLM is disabled.
- No exceptions are raised if the annotator is unable to process a test due to the LLM being disabled.
- The annotator's behavior remains consistent across different test configurations and environments.
- The `test_skips_if_disabled` function does not introduce any new test cases or dependencies that could cause issues.
- The annotator's functionality is still available when the LLM is enabled, as expected.
- The test does not rely on external state or variables that may change unexpectedly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	2 lines (ranges: 45-46)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: To prevent the annotator from attempting to annotate data when it cannot connect to the provider.

Key Assertions:

- mock_provider.is_available() returns False
- mock_provider.get_provider().get_data() raises an exception
- mock_provider.get_provider().get_data() does not contain any annotations
- mock_provider.get_provider().get_data() is not a list of dictionaries
- mock_provider.get_provider().get_data() contains only strings or None values
- mock_provider.is_available() returns False after the annotation process has been 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/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the annotator reports progress and the first error in concurrent mode.

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

Key Assertions:

- The function should append a message indicating the number of tasks completed (2) to the `progress_msgs` list.
- The function should append a message indicating that an error occurred ('first error') to the `progress_msgs` list.
- The function should append messages indicating progress for each task ('Processing 1 test(s)' and 'Processing 3 test(s)') to the `progress_msgs` list.
- The function should not report any errors when annotating in parallel (i.e., failures == 0).
- The function should report an error as a string ('first error') in the `progress_msgs` list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: The test verifies that the annotator waits if the rate limit interval has not elapsed after a sequential annotation task.

Why Needed: This test prevents regression where the annotator does not wait for the rate limit interval to elapse after a sequential annotation task, potentially leading to incomplete annotations or errors.

Key Assertions:

- The `time.sleep` function is called with an argument of `1.0s` (the rate limit interval).
- The `time.monotonic` function returns a value greater than `1.0s`, indicating that the elapsed time exceeds the rate limit interval.
- The `time.sleep` function is not called when the elapsed time is less than or equal to `0.1s`.
- The `time.monotonic` function returns a value less than or equal to `0.1s`, indicating that the elapsed time is within the allowed range.
- The `time.sleep` function is only called with an argument of `1.0s` when the rate limit interval has not elapsed, ensuring that the annotator waits for the interval to elapse before proceeding.
- If the rate limit interval had elapsed, the `time.sleep` function would have been called without any arguments, indicating a potential regression in the behavior of the annotator.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Should report progress for cached tests.

Why Needed: Prevents regression when caching tests and annotating them.

Key Assertions:

- The `get_provider` method of the LLMAnnotator is called with a mock provider.
- The `LlmCache` instance is created with a mock cache.
- The `ContextAssembler` class is patched to return a mock assembly.
- Any progress messages containing '(cache): test_cached' are appended to the list.
- The `get_provider` method of the LLMAnnotator returns True for the mock provider.
- The `LlmCache` instance has a cached annotation for the test.
- The `ContextAssembler` class correctly assembles the test with the correct source and None value.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 regression by ensuring that the annotator does not attempt to annotate tests when the provider is unavailable.

Key Assertions:

- mock_provider.is_available.return_value == False
- assert captured.out.contains('not available. Skipping annotations')
- annotate_tests(tests, config) should raise an exception or return immediately

COVERAGE

src/pytest_llm_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 the `_parse_response` method returns an error when a malformed JSON is provided.

Why Needed: This test prevents a potential bug where the function `extract_json_from_response` may incorrectly parse valid JSON due to missing or invalid braces.

Key Assertions:

- annotation.error == 'Failed to parse LLM response as JSON'
- response != '{ invalid json }'
- provider._parse_response(response) is not None
- provider._parse_response(response).error == 'Failed to parse LLM response as JSON'
- provider._parse_response(response).message == 'Invalid JSON: expected property '
- provider._parse_response(response).type == 'JSONDecodeError'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	10 lines (ranges: 52-53, 186-187, 190-191, 194-195, 220-221)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the `test_base_parse_response_non_string_fields` function correctly handles non-string fields in the response data.

Why Needed: To prevent a potential bug where the function does not handle non-string fields in the response data, and instead raises an error or returns incorrect results.

Key Assertions:

- The `scenario` attribute of the parsed annotation should match the provided value (123).
- The `why_needed` list should contain the expected field ('list').
- All elements in the `key_assertions` list should be present and match their corresponding values in the 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: Verify that the `get_gemini_provider` function returns a `GeminiProvider` instance.

Why Needed: Prevents regression in case of changes to the `Config` class or its dependencies.

Key Assertions:

- The returned value is an instance of `GeminiProvider`.
- The returned value has the correct type (`GeminiProvider`).
- No exceptions are raised when calling `get_gemini_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	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: Tests for the `get_provider` function when an unknown LLM provider is specified.

Why Needed: This test prevents a ValueError from being raised when an invalid LLM provider is provided to the `get_provider` function.

Key Assertions:

- The `get_provider` function should raise a `ValueError` with the message 'Unknown LLM provider: invalid'.
- The error message should contain the string 'invalid' (case-insensitive).
- The error message should not be empty.
- The `pytest.raises` context manager should be used to catch the `ValueError`.
- The `match` attribute of the error message should match the expected pattern ('Unknown LLM provider: invalid').
- The `Config` object passed to `get_provider` should have a 'provider' key with value '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	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: Verifies that the `get_litellm_provider` function returns a LitELLMProvider instance.

Why Needed: Prevents a potential issue where an incorrect or incompatible provider is returned for litellm.

Key Assertions:

- The function `get_provider(config)` should return an instance of `LiteLLMProvider`.
- The `isinstance(provider, LiteLLMProvider)` assertion should be true.
- The `provider` variable should hold an instance of `LiteLLMProvider` after calling `get_provider(config)`.
- An error message or indication that the provider is not compatible with litellm should not be returned.
- The function `get_provider(config)` should handle cases where no suitable provider is found.
- A meaningful exception message should be raised when an incompatible provider is encountered.
- The function `get_provider(config)` should return a valid provider instance for all possible configurations.
- An assertion error or indication of failure should not occur if the provider is correctly configured.

COVERAGE

src/pytest_llm_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 the `get_noop_provider` function returns a `NoopProvider` instance when no provider is specified.

Why Needed: This test prevents a regression where the `get_noop_provider` function fails to return an instance of `NoopProvider` when no provider is provided.

Key Assertions:

- The function `get_provider(config)` returns an instance of `Config` with a 'provider' attribute set to 'none'.
- The variable `provider` is assigned the value of `get_provider(config)`, which is an instance of `NoopProvider`.
- The assertion `assert isinstance(provider, NoopProvider)` passes because `get_provider(config)` returns an instance of `Config` with a 'provider' attribute set to 'none'.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The returned value is an instance of `OllamaProvider`.
- The `provider` attribute of the returned value has the correct value.
- No exception is raised when calling `get_provider(config)`.
- The `config` object passed to `get_provider` is not empty.
- The `config` object passed to `get_provider` does not contain a 'provider' key.
- The `provider` attribute of the returned value is set to a valid provider name ('ollama').

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the LlmProvider provides a single instance of the _check_availability method when available caches are available.

Why Needed: This test prevents regression in case multiple providers are used with available caches.

Key Assertions:

- The provider is not None and its checks attribute is not zero.
- The provider's is_available() method returns True for both instances.
- The provider's checks attribute is equal to the number of times it was called.
- The _check_availability method is called once when available caches are available.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 configuration when no custom model is specified.

Why Needed: This test prevents a potential bug where the 'get_model_name' method returns an incorrect default model name if no custom model is provided in the configuration.

Key Assertions:

- The 'model' attribute of the 'Config' object should be set to 'test-model'.
- The 'provider' object should have a 'get_model_name' method that returns the value of the 'model' attribute of the 'Config' object.
- The 'provider.get_model_name()' call should return 'test-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	3 lines (ranges: 52-53, 136)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The `get_rate_limits` method of the `ConcreteProvider` class returns a default value of `None` when no configuration is provided.

Why Needed: This test prevents a potential bug where the rate limits are not set to their default values when no configuration is provided.

Key Assertions:

- The `get_rate_limits()` method should return `None` if no configuration is provided.
- The `rate_limits` attribute of the `ConcreteProvider` class should be an empty list or dictionary.
- The `rate_limits` attribute of the `ConcreteProvider` class should not contain any values.
- The `rate_limits` attribute of the `ConcreteProvider` class should have a length of 0.
- The `rate_limits` attribute of the `ConcreteProvider` class should be an empty dictionary.
- The `rate_limits` attribute of the `ConcreteProvider` class should not contain any values.
- The `rate_limits` attribute of the `ConcreteProvider` class should have a length of 0.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: Verify that the `is_local()` method returns `False` when the default is set to `None`.

Why Needed: Prevents a potential bug where the default value of `True` for `is_local()` would be misinterpreted as `True`.

Key Assertions:

- provider.is_local() == False
- provider.is_local() != True
- config.default_is_local_defaults is None
- config.default_is_local_defaults not equal to True
- provider.is_local() in [False, True]
- provider.is_local() not in [True, False]
- config.default_is_local_defaults == 'None'
- config.default_is_local_defaults != 'None'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 147)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The 'hash_source' function is used to generate a consistent hash for the given source code.

Why Needed: This test prevents a potential issue where different versions of the same source code produce different hashes, leading to inconsistent caching.

Key Assertions:

- assert hash_source(source) == hash_source(source)
- assert len(hash_source(source)) == len(set(hash_source(source)))
- assert hash_source('def test_foo(): pass') == hash_source('def test_foo(): pass')

COVERAGE

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

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

AI ASSESSMENT

Scenario: Testing the `hash_source` function with different source code.

Why Needed: This test prevents a bug where identical source code produces the same hash value, potentially leading to unexpected behavior or errors in subsequent tests.

Key Assertions:

- The two source strings should have different hashes.
- The `hash_source()` function should return a different hash for each input string.
- The `hash_source()` function should raise an error if the input strings are identical.
- The `hash_source()` function should not produce the same hash value for the same input string and different source code.
- Different source code should produce different hashes even when the inputs are the same.
- The `hash_source()` function should be able to distinguish between different source codes based on their content.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify the length of the generated hash.

Why Needed: Prevent a potential issue where the hash length is not consistent across different test runs.

Key Assertions:

- The hash should be exactly 16 characters long.
- The hash should have a unique value for each input.
- The hash should not be shorter than 8 characters (the minimum required by Python's built-in hash function).
- The hash should not be longer than 32 characters (the maximum allowed by Python's built-in hash function).

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 clearing cache after adding entries.

Why Needed: To prevent a potential bug where the cache grows indefinitely and consumes excessive memory.

Key Assertions:

- The `clear` method should remove all cache entries.
- The `get` method for a specific key should return `None` if it does not exist.
- The number of remaining cache entries after clearing should be 2.
- A new entry should be added to the cache with no associated annotation.
- An existing entry should be removed from the cache with an invalid annotation.
- The cache directory should still exist and contain the expected files after clearing.
- No error should be raised if the cache is empty.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that annotations with errors are not cached.

Why Needed: This test prevents a bug where error annotations are incorrectly cached, leading to incorrect results when trying to access them later.

Key Assertions:

- The cache should not store the annotation for 'test::foo' with key 'abc123' and value 'error: API timeout'.
- The cache should return None when trying to retrieve the annotation for 'test::foo' with key '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 that `get` method returns None for missing entries in the cache.

Why Needed: Prevents a bug where an entry with no corresponding key is not properly removed from the cache.

Key Assertions:

- The function should return `None` when trying to retrieve a non-existent key.
- The `get` method should raise a `KeyError` exception when called on a missing key.
- The cache should be updated with the correct entry for the given key.
- The test should verify that the expected value is not retrieved from the cache.
- The function should handle cases where multiple keys are missing simultaneously.
- The cache should be properly cleaned up after each test run to prevent memory leaks.

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 that annotations are stored and retrieved correctly from the cache.

Why Needed: Prevents bypass attacks by ensuring that the annotation is stored in the cache before it can be used.

Key Assertions:

- Check if the annotation is set with the correct key
- Check if the annotation's scenario matches the expected value
- Check if the annotation's confidence matches the expected value
- Verify that a valid annotation is retrieved from the cache

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 CollectionError objects have the correct nodeID and message attributes.

Why Needed: This test prevents a potential bug where CollectionError objects are incorrectly structured, leading to incorrect assertions or errors in subsequent tests.

Key Assertions:

- The nodeID attribute of the error object is set correctly.
- The message attribute of the error object is set correctly.
- The nodeID value matches the expected string 'test_bad.py'.
- The message value matches the expected string '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 'test_get_collection_errors_initially_empty' verifies that an empty collection is returned when the test collector is created with a valid configuration.

Why Needed: This test prevents a potential regression where an empty collection is returned unexpectedly when the test collector is initialized with a valid configuration.

Key Assertions:

- The function `get_collection_errors()` should return an empty list.
- The function `get_collection_errors()` should not raise any exceptions.
- The function `get_collection_errors()` should not throw any errors or warnings.
- The function `get_collection_errors()` should only return the collection errors.
- The function `get_collection_errors()` should not return any errors when the collection is empty.
- The function `get_collection_errors()` should not raise an exception when the collection is empty.
- The function `get_collection_errors()` should not throw a warning when the collection is empty.
- The function `get_collection_errors()` should only log warnings when the collection is empty.

COVERAGE

src/pytest_llm_report/collector.py	15 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 285)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the default value of llm_context_override is None when it's not provided.

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

Key Assertions:

- llm_context_override is None for TestCaseResult.nodeid='test.py::test_foo' and outcome='passed'
- llm_context_override is None for TestCaseResult.nodeid='test.py::test_foo' and outcome='failed'
- llm_context_override is not None for TestCaseResult.nodeid='test.py::test_foo' and outcome='skipped'
- llm_context_override is not None for TestCaseResult.nodeid='test.py::test_foo' and outcome='aborted'
- llm_context_override is not None for TestCaseResult.nodeid='test.py::test_foo' and outcome='not_found'
- llm_context_override is not None for TestCaseResult.nodeid='test.py::test_foo' and outcome='unknown'
- llm_context_override is not None for TestCaseResult.nodeid='test.py::test_foo' and outcome='timeout'
- llm_context_override is not None for TestCaseResult.nodeid='test.py::test_foo' and outcome='not_supported'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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 the default value of llm_opt_out for LLM optimization out.

Why Needed: Prevents regression in LLM optimization behavior when default is set to False.

Key Assertions:

- The `llm_opt_out` attribute of a TestCaseResult instance should be `False`.
- A TestCaseResult instance with `nodeid` 'test.py::test_foo' and `outcome` 'passed' should have `llm_opt_out` set to `False`.
- When the default value of `llm_opt_out` is set to False, a TestCaseResult instance without any other attributes (like `nodeid`, `outcome`) should also have `llm_opt_out` 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	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 output capture feature is not enabled by default.

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

Key Assertions:

- config.capture_failed_output should be set to False when no capture is specified.
- config.capture_enabled should be set to False when no capture is specified.
- The output of captured messages should not be displayed in the console.
- No error message should be printed when capturing fails.
- The test should fail if config.capture_failed_output is True and config.capture_enabled is True.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 `TestCollectorOutputCapture` class is configured to capture output with a default maximum character count of 4000.

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

Key Assertions:

- config.capture_output_max_chars
- assert config.capture_output_max_chars == 4000
- assert isinstance(config.capture_output_max_chars, int)
- assert config.capture_output_max_chars >= 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)`

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

AI ASSESSMENT

Scenario: Test 'tests/test_collector.py::TestCollectorXfailHandling::test_xfail_failed_is_xfailed' verifies that failed xfail tests are recorded as xfailed.

Why Needed: This test prevents regression in handling xfail failures, ensuring that the collector correctly records and reports on failed xfail tests.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	`36 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-118, 124, 127, 140, 155-159, 163, 167, 171, 209-210, 212, 216, 227-228, 230-234, 238)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 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 passes are correctly recorded as xpassed in the report.

Why Needed: This test prevents regression where an unexpected pass is not properly recorded as a failure.

Key Assertions:

- result.outcome should be 'xpassed' when report.wasxfail == 'expected failure'
- result.outcome should be 'xpassed' when report.failed == False and report.skipped == False
- report.wasxfail should be 'expected failure' when result.outcome is 'xpassed'

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 creation of a Collector instance with an empty configuration.

Why Needed: This test prevents a potential bug where the Collector does not initialize correctly when given an empty configuration.

Key Assertions:

- The `results` attribute is set to an empty dictionary.
- The `collection_errors` list is empty.
- The `collected_count` attribute is set to 0.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The list of node IDs returned by `get_results()` should be in ascending order.
- The list of node IDs returned by `get_results()` should contain all nodes from both tests.
- The list of node IDs returned by `get_results()` should not contain any duplicate node IDs.
- The first node ID in the list should be 'a_test.py::test_a'.
- The second node ID in the list should be 'z_test.py::test_z'.

COVERAGE

src/pytest_llm_report/collector.py	15 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the collector might incorrectly report the number of collected or deselected items when the collection is finished.

Key Assertions:

- The `collected_count` attribute should be updated with the correct count after calling `handle_collection_finish`.
- The `deselected_count` attribute should be updated with the correct count after calling `handle_collection_finish`.
- The number of collected items should match the expected value (3) before and after calling `handle_collection_finish`.
- The number of deselected items should match the expected value (1) before and after calling `handle_collection_finish`.
- The total number of collected and deselected items should be equal to the sum of their counts before and after calling `handle_collection_finish`.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The `collector.handle_runtest_logreport(report)` call should not modify the `result.captured_stdout` attribute.
- The `results` dictionary in the test case `t` should still contain a key named 'output' with value 'failed'.
- The `collector.results` dictionary in the test case `t` should have an empty list for the key 'skipped'.

COVERAGE

src/pytest_llm_report/collector.py	36 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-118, 124, 127-128, 130, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the `test_capture_output_stderr` function captures stderr correctly.

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

Key Assertions:

- The `captured_stderr` attribute of the `TestCaseResult` object should be set to the expected value.
- The `report.capstderr` method should return the expected error message.
- The `collector._capture_output(result, report)` function should record the captured stderr correctly.

COVERAGE

src/pytest_llm_report/collector.py	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264, 268-269)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `captured_stdout` attribute of the `TestCaseResult` object should contain the expected output.
- The `report.capstdout` method should be called with the correct value.
- The `report.capstderr` method should be called with an empty string.
- The captured stdout should not be empty after calling `_capture_output`.
- The captured stderr should be empty after calling `_capture_output`.
- The `result.captured_stdout` attribute should contain the expected output.

COVERAGE

src/pytest_llm_report/collector.py	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264-265, 268)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the `test_capture_output_truncated` function truncates output exceeding the specified maximum characters.

Why Needed: This test prevents a potential bug where the collector fails to truncate output exceeding the max chars limit, leading to incorrect results or unexpected behavior.

Key Assertions:

- The captured stdout length is less than or equal to the `capture_output_max_chars` value.
- The captured stderr length is zero.
- The captured stdout contains only characters up to the specified maximum length.
- The captured stderr does not contain any characters.

COVERAGE

`src/pytest_llm_report/collector.py`	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264-265, 268)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test `test_create_result_with_item_markers` verifies that the collector extracts item markers correctly.

Why Needed: This test prevents a potential regression where the collector might not extract all required item markers.

Key Assertions:

- item.callspec.id should be set to 'param1' when calling get_closest_marker('llm_opt_out')
- item.get_closest_marker('llm_context_override') should return a mock object with 'complete' as argument
- result.param_id should match the value of `item.callspec.id`
- result.llm_opt_out should be set to True based on the marker name
- result.llm_context_override should match the expected value from get_closest_marker('llm_context_override')
- result.requirements should contain 'REQ-1' and 'REQ-2' as strings

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 to handle ReprFileLocation behavior.

Why Needed: This test prevents a potential crash when handling ReprFileLocation in error reports.

Key Assertions:

- The `report.longrepr.__str__.return_value` is set to 'Crash report'.
- The `_extract_error` method returns the expected string 'Crash report'.

COVERAGE

src/pytest_llm_report/collector.py	22 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test the `_extract_error` method of `TestCollector` to verify it returns a string longrepr when an error occurs.

Why Needed: Prevents regression in case of unexpected error during report extraction.

Key Assertions:

- The `_extract_error` method should return 'Some error occurred' as the extracted error string.
- The `report.longrepr` attribute should be set to 'Some error occurred'.
- The `collector._extract_error(report)` expression should evaluate 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: Test that the `_extract_skip_reason` method returns `None` when no longrepr is provided.

Why Needed: Prevents a potential bug where the method does not handle cases where `longrepr` is `None`.

Key Assertions:

- The `_extract_skip_reason` method should return `None` for an empty or `None` `report.longrepr`.
- The `_extract_skip_reason` method should raise a `ValueError` when `report.longrepr` is `None`.
- The `_extract_skip_reason` method should not attempt to extract the skip reason from an empty report.
- The `_extract_skip_reason` method should return `None` for a `report` object with no attributes.
- The `_extract_skip_reason` method should raise an error when called on a non-report object.
- The `_extract_skip_reason` method should not attempt to extract the skip reason from a report that does not have a `longrepr` attribute.
- The `_extract_skip_reason` method should return `None` for a report with a `longrepr` attribute set to an empty 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, 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 `test_extract_skip_reason_string` verifies that the `_extract_skip_reason` method returns a string as expected when given a `report` object.

Why Needed: This test prevents a potential bug where the `longrepr` attribute of the `report` object is not correctly extracted and returned by the `_extract_skip_reason` method.

Key Assertions:

- The value of `report.longrepr` should be 'Just skipped'.
- The string 'Just skipped' should be returned as the skip reason.
- If the report does not have a `longrepr` attribute, an error should be raised with a meaningful message.
- If the `report` object is not a `MagicMock` instance, the test should fail with a clear error message.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250-251)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that `collectors._extract_skip_reason` extracts skip message from a tuple containing file, line and message.

Why Needed: Prevents regression where a test fails due to incorrect handling of tuples with multiple values.

Key Assertions:

- The `report.longrepr` attribute is not converted to a string before being passed to `_extract_skip_reason`.
- The `collectors._extract_skip_reason(report)` function should correctly extract the skip message from the tuple containing file, line and message.
- The expected output of `_extract_skip_reason` is correct for all cases where it's applicable.

COVERAGE

`src/pytest_llm_report/collector.py`	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250-251)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `handle_collection_report` method does not record errors when they occur during collection.

Key Assertions:

- The `collection_errors` list should contain exactly one item with `nodeid='test_broken.py'` and `message='SyntaxError'`.
- The error message should be 'SyntaxError'.
- The node ID of the error should match 'test_broken.py'.

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 'Should handle rerun attribute.' verifies that the collector correctly handles rerun attributes in runtest logs.

Why Needed: This test prevents a potential regression where the collector might not handle rerun attributes correctly, leading to incorrect results or unexpected behavior.

Key Assertions:

- res.rerun_count should be equal to 1 (one rerun)
- res.final_outcome should be 'failed' (the final outcome of the runtest)
- collector.results['t::r'] should contain the expected result (in this case, a dictionary with 'rerun_count' and 'final_outcome')

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 `handle_runtest_setup_failure` method records a setup error in the report.

Why Needed: This test prevents a potential regression where a setup failure is not properly recorded in the report, potentially leading to incorrect analysis or reporting.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the collector does not record errors when the teardown function fails after passing the runtest.

Key Assertions:

- assert res.outcome == 'error'
- assert res.phase == 'teardown'
- assert res.error_message == 'Cleanup failed'
- assert call_report.failed is True
- assert teardown_report.failed is True

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 GeminiProvider's parsing of preferred models for edge cases, including an empty list and a specific model.

Why Needed: This test prevents regression in case the 'GeminiProvider' class is modified to handle edge cases where the preferred models are not provided.

Key Assertions:

- The function `provider._parse_preferred_models()` returns a list of strings containing the specified models.
- The function `provider._parse_preferred_models()` returns an empty list when no preferred models are provided.
- The function `provider._parse_preferred_models()` returns a list containing 'All' when the model is set to 'All'.
- The function `provider._parse_preferred_models()` does not return any values when the model is None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	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 prevents over and under token limits when recording tokens but not requests.

Why Needed: This test prevents a potential bug where the rate limiter allows excessive token usage without triggering an error, potentially leading to performance issues or security vulnerabilities.

Key Assertions:

- assert limiter.next_available_in(60) > 0
- assert limiter.next_available_in(10) == 0
- assert limiter.record_tokens(50) is None
- assert len(limiter.tokens) < 100
- assert len(limiter.requests) < 10

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	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 `models_to_dict` method returns accurate coverage percentages for SourceCoverageEntry instances with varying numbers of statements, covered and missed lines.

Why Needed: This test prevents regression in coverage calculation when dealing with SourceCoverageEntry objects with different numbers of statements, covered, or missed lines.

Key Assertions:

- The `coverage_percent` attribute is correctly set to the specified value for each SourceCoverageEntry instance.
- The `covered_ranges` and `missed_ranges` attributes are correctly formatted as strings within their respective keys in the assertion.
- The `duration` attribute is correctly set to 1.0 seconds for each RunMeta instance, regardless of its start time or duration.
- For LlmAnnotation instances with an error message 'timeout', the corresponding `error` key should contain 'timeout' as expected.
- When creating a new SourceCoverageEntry object with incorrect data (e.g., missing statements), the `to_dict()` method should raise an AssertionError indicating that the coverage percentage is not 50.0%.
- The `to_dict()` method should correctly handle cases where the number of covered or missed lines exceeds the total number of lines in the file.
- For RunMeta instances with a start time before the current time, the corresponding `duration` key should be set to 1.0 seconds 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	46 lines (ranges: 71-78, 104-107, 109, 111-113, 115, 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Testing the creation of a CoverageMapper instance with an empty configuration.

Why Needed: This test prevents a potential bug where a CoverageMapper created with an invalid or missing configuration would not throw an error, but instead silently initialize and return a non-configured object.

Key Assertions:

- The `config` attribute of the `CoverageMapper` instance is set to the provided `Config` object.
- The `warnings` attribute of the `CoverageMapper` instance is initialized with an empty list.
- An assertion error would be raised if the `config` attribute was not set or was missing in the `CoverageMapper` 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/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: Verify the `get_warnings` method returns a list of warnings.

Why Needed: Prevents a potential bug where the function does not return a list of warnings as expected.

Key Assertions:

- The `get_warnings()` method should return a list of warnings.
- The `get_warnings()` method should be able to handle any configuration or error scenarios.
- The `get_warnings()` method should raise an exception if no warnings are found, as per the test requirements.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 verifies that the `map_coverage` method returns an empty dictionary when no coverage file exists.

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

Key Assertions:

- The `Path.exists` mock should return False for the given input.
- The `glob.glob` mock should return an empty list for the given input.
- The `map_coverage` method should return an empty dictionary.
- At least one warning should be emitted by the test.
- The number of warnings emitted 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/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the `CoverageMapper` extracts 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 `_extract_nodeid` method should return the correct node ID for each phase.
- The `_extract_nodeid` method should not return any node IDs when `include_phase=all`.
- The `_extract_nodeid` method should include all phases in the coverage map.
- The coverage map should have all phases included if `include_phase=all` is used.
- The coverage map should exclude no phases if `include_phase=all` is used.
- The node IDs returned by `_extract_nodeid` should match the expected node IDs for each 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	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_extract_nodeid_filters_setup` test case extracts node IDs correctly when `include_phase=run`.

Why Needed: This test prevents a potential bug where setup phase nodes are incorrectly filtered out during coverage analysis.

Key Assertions:

- The function `_extract_nodeid` in the `CoverageMapper` class should extract node IDs from the given string.
- The extracted node ID should be `None` when the input string contains 'setup'.
- The `include_phase` parameter in the `Config` object should match the expected phase ('run') for this test case.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 extracts the correct node ID from a run phase context.

Why Needed: This test prevents coverage regression by ensuring that the `extract_nodeid` method correctly identifies nodes in the run phase of the code.

Key Assertions:

- The extracted node ID matches the expected value (`test.py::test_foo`).
- The node ID is present in the context string (`test.py::test_foo|run`).
- The node ID is not empty or null.
- The node ID does not contain any whitespace characters.
- The node ID does not start with a digit.
- The node ID does not contain any special characters (e.g., !, @, #, $, 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/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 extract all contexts for full logic coverage.

Why Needed: Prevents regression in coverage reporting when the test is run with `omit_tests_from_coverage=True`.

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: The test verifies that the `CoverageMapper` correctly handles data with no test contexts by returning an empty dictionary.

Why Needed: This test prevents a bug where the `CoverageMapper` returns incorrect results when given data without any test contexts.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test extracts node ID variants for different phases.

Why Needed: Prevents regression in coverage analysis by ensuring the correct extraction of node IDs based on phase.

Key Assertions:

- The function `_extract_nodeid` correctly identifies the target node ID when a phase is specified.
- The function `_extract_nodeid` ignores the target node ID when no phase is specified.
- The function `_extract_nodeid` handles cases without phases (e.g., `test.py::test_no_phase`) 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/coverage_map.py	14 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231-234, 236, 239)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the function `test_load_coverage_data_no_files` raises a warning when no coverage files exist.

Why Needed: This test prevents a potential bug where the function does not raise an error or warning for scenarios with no coverage data.

Key Assertions:

- The function `_load_coverage_data()` returns `None` when no coverage files are found.
- The list of warnings is expected to contain exactly one entry with code 'W001'.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the CoverageMapper class fails to handle errors when loading coverage data from corrupted or invalid files.

Key Assertions:

- The function should return None when attempting to load coverage data from a corrupt .coverage file.
- Any warnings generated by the mapper should contain the message 'Failed to read coverage data'.
- The function should not attempt to load coverage data from a valid .coverage file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	17 lines (ranges: 44-45, 72-73, 83, 86, 88, 92, 94-96, 107-111, 114)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the CoverageMapper does not update its internal data structures when loading coverage data with parallel files from xdist.

Key Assertions:

- The `update` method of the `CoverageData` class should be called at least twice when loading coverage data with parallel files.
- The `update` method of the `CoverageData` class should not be called zero times when loading coverage data with parallel files.
- The `update` method of the `CoverageData` class should call the `__call__` method on the mock instances correctly when loading coverage data with parallel files.
- The `update` method of the `CoverageData` class should update its internal state correctly when loading coverage data with parallel files.
- The `update` method of the `CoverageData` class should not raise an exception when loading coverage data with parallel files.
- The mock instances returned by the `patch` function should be different from each other when loading coverage data with parallel files.
- The mock instance for `mock_parallel_data1` should have a call count greater than 0 when loading coverage data with parallel files.
- The mock instance for `mock_parallel_data2` should not have a call count greater than 0 when loading coverage data with parallel 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 that the `map_coverage` method returns an empty dictionary when _load_coverage_data returns None.

Why Needed: Prevents a potential bug where the test fails with an unexpected error or incorrect result when there is no coverage data.

Key Assertions:

- The `_load_coverage_data` function should return `None` when called with no arguments.
- The `map_coverage` method should return an empty dictionary when passed an empty dictionary as input.
- The `map_coverage` method should not raise any exceptions when called with a non-empty dictionary 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/coverage_map.py	5 lines (ranges: 44-45, 58-60)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the coverage map does not include files with analysis errors.

Why Needed: To prevent coverage maps from including error-prone code, which can indicate a deeper issue in the application.

Key Assertions:

- mock_cov.analysis2.assert_called_once_with(mock_data)
- mock_data.measured_files.return_value == ['app.py']
- entries is an empty list after skipping files with errors

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	22 lines (ranges: 44-45, 243-244, 246-248, 250, 252-254, 259, 261, 263-268, 271, 299-300)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Test 'Should exercise all paths in map_source_coverage' verifies that the test covers all possible source code files.

Why Needed: This test prevents regression by ensuring that the coverage mapper exercises all possible source code files, even if they are not directly analyzed by analysis2.

Key Assertions:

- The function `map_source_coverage` returns a list of entries where each entry contains information about a file's coverage.
- Each entry in the returned list has the following properties: `file_path`, `statements`, `covered`, `missed`, and `coverage_percent`.
- The `coverage_percent` property is calculated by dividing the number of statements covered by the total number of statements in the file, then multiplying by 100.
- The test verifies that each entry has a unique `file_path`, `statements`, `covered`, `missed`, and `coverage_percent`.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the `make_warning` factory function with a valid warning code and message.

Why Needed: Prevents a potential warning that occurs when no .coverage file is found for a test.

Key Assertions:

- The function returns an instance of WarningCode.W001_NO_COVERAGE with the correct code.
- The `message` attribute contains the expected string 'No .coverage file found'.
- The `detail` attribute matches the provided value 'test-detail'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Tests that warning codes have correct values.

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

Key Assertions:

- {'name': 'Correct value for W001_NO_COVERAGE', 'expected_value': 'W001'}
- {'name': 'Correct value for W101_LLM_ENABLED', 'expected_value': 'W101'}
- {'name': 'Correct value for W201_OUTPUT_PATH_INVALID', 'expected_value': 'W201'}
- {'name': 'Correct value for W301_INVALID_CONFIG', 'expected_value': 'W301'}
- {'name': 'Correct value for W401_AGGREGATE_DIR_MISSING', 'expected_value': 'W401'}

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test Warning.to_dict() method to ensure it returns a valid dictionary with required keys.

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

Key Assertions:

- The 'code' key should be present and have the correct value ('W001').
- The 'message' key should be present and have the correct value ('No coverage').
- The 'detail' key should be present and have the correct value ('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 makes sure a warning with standard message is created when known code is used.

Why Needed: Prevents regression where unknown code may cause unexpected warnings.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test Make Warning: Unknown Code

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

Key Assertions:

- The function `make_warning` does not raise an exception when given an unknown warning code.
- The function `make_warning` uses the correct fallback message for unknown code.
- The function `make_warning` restores the original message after using it to make a warning.
- The function `make_warning` correctly handles missing warning codes by returning the old message.
- The function `make_warning` does not raise an exception when given an invalid warning code.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test Make Warning with Detail: Verifies that a warning is created with the correct code and detail.

Why Needed: This test prevents a potential bug where a warning is not properly created when an invalid configuration value is provided, potentially leading to unexpected behavior or errors.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that all enum values are strings and start with 'W' to prevent WarningsCodes from being used as non-string codes.

Why Needed: This test prevents a potential bug where WarningCode enum values could be misused as non-string codes, leading to incorrect warnings.

Key Assertions:

- assert isinstance(code.value, str) checks if the value of each code is indeed a string.
- assert code.value.startswith('W') checks if all enum values start with 'W' and are therefore valid warning codes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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 class can be serialized to a dictionary without including detailed information.

Why Needed: This test prevents a potential issue where the warning message is not properly represented in the serialized data.

Key Assertions:

- The 'code' key should contain the warning code.
- The 'message' key should contain the warning message.
- Both keys should match the expected values ('W001' and 'No coverage', respectively).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	5 lines (ranges: 70-72, 74, 76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test the warning_to_dict method with detailed information.

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

Key Assertions:

- The `to_dict()` method of Warning objects returns a dictionary with the correct keys ('code', 'message', and 'detail').
- The 'detail' key in the returned dictionary contains the expected string value ('Check setup').
- The 'message' key in the returned dictionary contains the expected string value ('No coverage').
- The 'code' key in the returned dictionary contains the correct string value ('W001').

COVERAGE

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

Why Needed: Prevents a potential bug where the function incorrectly returns True for non-python files.

Key Assertions:

- The function should return False when given a file with a non-.py extension (e.g. 'foo/bar.txt')
- The function should return False when given a file with a non-pyc extension (e.g. 'foo/bar.pyc')
- The function should not incorrectly report True for files with non-python extensions (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 `is_python_file` function correctly identifies .py files.

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

Key Assertions:

- The function should return True for files with names starting with '.py'.
- The function should raise an error or return False for files without names starting with '.py'.
- The function should handle file extensions other than '.py' correctly.
- The function should not incorrectly identify non-existent .py files as Python files.
- The function should be able to handle files with different casing in their names (e.g., 'Foo.py', '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/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 a potential bug where the test fails if the test directory is not in the same Python package as the test file.

Key Assertions:

- The `make_relative` function should return the expected file path.
- The file path returned by `make_relative` should be an absolute path relative to the test directory.
- The parent directory of the original file path should be created if it does not exist.
- The `touch` method should not raise an exception if the file already exists.
- The `mkdir` method should create the parent directories if they do not exist.
- The `file_path.parent.mkdir(parents=True, exist_ok=True)` line should work correctly even if the test directory is not in the same package as the test file.
- The `make_relative` function should be able to handle cases where the original file path is a relative 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	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: test returns normalized with no base

Why Needed: Prevents regression where the function does not normalize paths correctly without a base.

Key Assertions:

- The function `make_relative` should return the original path when there is no base.
- The function `make_relative` should handle cases where the input is an absolute path.
- The function `make_relative` should correctly normalize the resulting relative path.
- The function `make_relative` should not modify the original path.
- The function `make_relative` should raise a meaningful error when given invalid input.

COVERAGE

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

Why Needed: This test prevents a potential bug where the `normalize_path` function incorrectly normalizes paths that are already in their normalized form.

Key Assertions:

- The output of `normalize_path('foo/bar')` should be `'foo/bar'`.
- The input path 'foo/bar' is already normalized, so it should not be modified by the `normalize_path` function.
- If the input path is already in its normalized form, then the output should also be in its normalized form.
- If the input path has leading or trailing whitespace, then the output should have the same leading/trailing whitespace.
- If the input path contains any invalid characters (e.g. `/./`, `:`, etc.), then the output should not contain any such characters.
- The function should handle cases where the input path is a relative path (i.e. it does not start with a slash).
- The function should handle cases where the input path is an absolute path (i.e. it starts with a slash).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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 a scenario where forward slashes are used in the path.

Why Needed: This test prevents regression when handling paths that contain forward slashes, which is necessary for Unix-based systems.

Key Assertions:

- The `normalize_path` function correctly converts backslashes to forward slashes in the input path.
- The resulting normalized path does not contain any backslashes.
- The function handles paths with multiple consecutive forward slashes correctly.
- The function preserves the original directory structure of the input path.
- The function is case-insensitive when handling paths that contain forward slashes.
- The function correctly handles paths with leading or trailing forward slashes.
- The function does not modify the original 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::TestNormalizePath::test_strips_trailing_slash 1ms 3

AI ASSESSMENT

Scenario: Verifies the normalization of a path with a trailing slash.

Why Needed: Prevents a potential issue where a file path with a trailing slash is not correctly normalized to remove it.

Key Assertions:

- The function `normalize_path` removes any leading or trailing slashes from the input path.
- If the input path does not start or end with a slash, the function should still return the original path.
- The function handles paths with multiple consecutive slashes correctly.
- It is not necessary to strip the trailing slash if it is already present in the input path.
- The function preserves the relative path information when stripping leading/trailing slashes.
- If the input path starts with a slash, the function should return the same path without stripping any additional slashes.
- The function does not modify the original path but returns a new normalized 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: Test verifies whether a path matches custom exclusion patterns.

Why Needed: This test prevents the test module from skipping paths that match custom patterns, ensuring consistent testing behavior.

Key Assertions:

- The 'tests/conftest.py' file should be skipped if it contains any of the excluded patterns ('test*').
- The 'src/module.py' file should not be skipped if it does not contain any of the excluded patterns ('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/fs.py	15 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116-117, 119-121, 123)

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

AI ASSESSMENT

Scenario: Verifies that the `should_skip_path` function returns `False` for a normal path.

Why Needed: Prevents a potential bug where the function incorrectly skips normal paths without checking their contents.

Key Assertions:

- The function should return `True` when given a normal path (`src/module.py`).
- The function should not return `False` when given a normal path (`src/module.py`).
- The function should raise an exception or return a specific value indicating that the path is skipped.
- The function should check the file contents before making a decision about skipping it.
- The function should handle cases where the path does not exist or is not accessible.
- The function should provide clear and consistent error messages for different scenarios.
- The function should be able to skip paths that are intended to be skipped (e.g., `__pycache__` 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	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: tests/test_fs.py::TestShouldSkipPath::test_skips_git verifies whether a .git directory should be skipped.

Why Needed: This test prevents the test from skipping non-.git directories, which could lead to unexpected behavior or errors.

Key Assertions:

- assert should_skip_path('.git/objects/foo') is True
- should not skip '.git' (expected False)
- should not skip '.git/objects/bar' (expected False)

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verifies that the `should_skip_path` function correctly skips `__pycache__` directories.

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

Key Assertions:

- The path `foo/__pycache__/bar.pyc` is not included in the list of paths to skip.
- The path `foo/__pycache__/bar.pyc` does not contain any executable code.
- The path `foo/__pycache__/bar.pyc` has a file extension that indicates it's a cache file (e.g. `.pyc`, `.pyc`).
- The `should_skip_path` function is correctly handling the case where the directory contains a non-executable file.
- The `should_skip_path` function is not including directories with non-ASCII characters in the list of paths to skip.
- The `should_skip_path` function is correctly skipping directories that are not Python modules (e.g. directories containing other types of files).
- The `should_skip_path` function is not incorrectly including directories with a file extension that indicates it's a cache file (e.g. `.pyc`, `.o`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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: test_skips_venv verifies whether the 'venv' directory is skipped by the function `should_skip_path`.

Why Needed: This test prevents a potential issue where the function `should_skip_path` incorrectly identifies 'venv' directories as being to be skipped, potentially leading to incorrect behavior in subsequent tests or code.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the rate limiter does not clear the request and token usage lists when a request is made in the past.

Key Assertions:

- The length of _request_times should be 0 after pruning.
- The length of _token_usage should be 0 after pruning.
- The _request_times list should contain only one element (the time of the past request).
- _request_times list should not contain any requests made in the future.
- The _token_usage list should contain two elements (the time and value of the past request) before pruning.
- The _token_usage list should be empty 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 the specified limit.

Why Needed: This test prevents a potential bug where multiple requests are made in quick succession, potentially overwhelming the server.

Key Assertions:

- The `next_available_in` method should return a value greater than 0.
- The `next_available_in` method should be less than or equal to 60.0 seconds.
- The limiter should not be unavailable after recording a request.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	26 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95, 97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the rate limiter prevents a regression when there are not enough tokens available.

Why Needed: This test verifies that the rate limiter does not allow too many requests to be made in a short period of time, preventing potential abuse.

Key Assertions:

- The next_available_in method returns a non-negative value (0) if there are no tokens left for the given number of requests.
- The _token_usage list is updated correctly after recording tokens and waiting for an available slot.
- The length of the _token_usage list remains unchanged at 2 even after multiple records and waits.

COVERAGE

src/pytest_llm_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: The test verifies that the `wait_for_slot` method of `_GeminiRateLimiter` sleeps for a specified amount of time.

Why Needed: This test prevents a potential issue where the rate limiter does not sleep when it should, potentially leading to unexpected behavior or errors in other parts of the application.

Key Assertions:

- The `wait_for_slot` method is called with the correct argument (1)
- The `time.sleep` mock object is called with the correct argument (1)
- The `time.sleep` mock object is not called before the first call to `wait_for_slot`
- The `time.sleep` mock object is called after the last call to `wait_for_slot`
- The `time.sleep` mock object is called within a reasonable amount of time (1 second)
- The rate limiter does not sleep when it should, potentially leading to unexpected behavior or errors in other parts of the application

COVERAGE

src/pytest_llm_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 rate limiter records zero tokens when no tokens are available.

Why Needed: This test prevents a potential bug where the rate limiter does not record tokens for an extended period, potentially leading to unexpected behavior or errors in downstream applications.

Key Assertions:

- The `_token_usage` attribute of the `limiter` object should be empty after calling `record_tokens(0)`.
- The `len(limiter._token_usage)` should be equal to 0 after calling `record_tokens(0)`.
- If no tokens are available, the rate limiter should not attempt to record any additional tokens.
- If an error occurs while recording tokens, it should not propagate to downstream applications.
- The rate limiter's `_token_usage` attribute should be reset to its initial state after calling `record_tokens(0)`.
- Calling `record_tokens(n)` with a non-zero value should not have any effect on the rate limiter's behavior.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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` 2ms 3

AI ASSESSMENT

Scenario: Verify that the test prevents a rate limit exceeded error when exceeding daily requests.

Why Needed: This test ensures that the Gemini Rate Limiter does not exceed the daily request limit, preventing an error from being raised.

Key Assertions:

- The limiter raises `_GeminiRateLimitExceeded` with the message `requests_per_day` when the daily limit is exceeded.
- The limiter records a request before raising the rate limit exceeded error.
- The limiter waits for a slot of at least 10 requests to be available after exceeding the 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/gemini.py`	24 lines (ranges: 32-34, 39-42, 45-46, 48-50, 58-60, 73, 76-78, 81-82, 84, 87-88)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the TPM fallback wait time is sufficient to prevent rate limiting when there are insufficient tokens available.

Why Needed: This test prevents a potential rate limiting issue where the TPM would fall back to waiting for an extended period of time if there were not enough tokens available.

Key Assertions:

- The `wait` variable should be greater than 0.
- The sum of `tokens_used` and `request_tokens` should be less than or equal to the rate limit (`10` tokens per minute).
- If `token_usage` is empty, then `limiter._seconds_until_tpm_available(now, 5)` should return a non-zero 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/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 593ms 6

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the RPM rate limit cooldown is not properly handled, leading to incorrect behavior or errors.

Key Assertions:

- The `models/gemini-pro` model should be present in the `_cooldowns` dictionary.
- The value of `models/gemini-pro` in the `_cooldowns` dictionary should be greater than 1000.0.
- The RPM rate limit cooldown handling should correctly handle the first call to `_GeminiRateLimitExceeded` with a retry after 0.1 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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 annotate method retries when rate limiting occurs and returns a valid annotation for the recovered scenario.

Why Needed: This test prevents regression in case of rate limiting, ensuring that the annotate method correctly handles retry attempts.

Key Assertions:

- The annotation returned by _annotate_internal is 'Recovered Scenario'.
- The mock_post call count is 2, indicating two successful retries.
- The scenario assertion passes for the recovered scenario.
- The mock_parse call count is 1, indicating only one parsing attempt.
- The mock_get and mock_post return status codes of 429 and 200 respectively, as expected.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that _annotate_internal returns LlmAnnotation correctly when _parse_response is called with the expected format.

Why Needed: This test prevents a potential regression where _parse_response might expect an incorrect format, leading to incorrect annotations.

Key Assertions:

- The annotation returned by _annotate_internal has the correct scenario.
- The annotation does not have any errors.
- _parse_response returns the expected response structure when called with the correct scenario and error.
- The annotation's scenario is correctly set to 'Success Scenario'.
- The annotation does not contain any error information.
- _build_prompt is mocked to avoid complex dependency, ensuring the test environment remains consistent.
- _annotate_internal calls _parse_response where it expects a specific format, allowing for correct annotations.
- Mocking _parse_response ensures that the test can focus on verifying the correctness of _annotate_internal without worrying about incorrect response formats.

COVERAGE

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

src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the availability check fails when environment variables are not set

Why Needed: Prevents a potential bug where the availability check returns False even if it's supposed to be available.

Key Assertions:

- provider._check_availability() is False
- provider._check_availability() is True

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 does not exceed the daily limit when no requests are made in a given time period.

Why Needed: The test prevents a potential bug where the rate limiter exceeds the daily limit and causes an error or unexpected behavior.

Key Assertions:

- limiter.record_request() should not be called before checking if there is an available slot for the next request.
- limiter.next_available_in(100) should return None when no requests are made in a given time period.
- The rate limiter's daily limit should not be exceeded even after multiple requests are made within the same time 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: Verify that the rate limiter does not block requests when there are available slots.

Why Needed: This test prevents a potential bug where the rate limiter blocks all requests for a short period, causing unexpected behavior or errors in downstream applications.

Key Assertions:

- The next_available_in method should return 0.0 after the first two requests.
- The next_available_in method should return 0.0 after the third request and before the wait time.
- The wait time should be between 0 and 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/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 hash collision bug where two different configurations with the same provider could have the same hash.

Key Assertions:

- The `compute_config_hash` function should return a different hash for `config1` and `config2`.
- The `compute_config_hash` function should not return the same hash when both `config1` and `config2` are from different providers.
- The `compute_config_hash` function should raise an exception if both `config1` and `config2` are from the same provider.
- The `compute_config_hash` function should be able to distinguish between `config1` and `config2` based on their provider.
- If `config1` is from provider 'ollama' and `config2` is from provider 'none', they should have different hashes.
- If both `config1` and `config2` are from provider 'none', the hash should be the same.
- The order of configuration providers does not affect the hash.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Tests that the computed hash is of length 16.

Why Needed: Prevents a potential issue where the hash might be too long for efficient storage or comparison purposes.

Key Assertions:

- The length of the computed hash should be exactly 16 characters.
- The hash value should not exceed 16 bytes (128 bits) in length.
- A hash with more than 16 characters should raise an error or indicate an issue.
- The hash is a valid SHA-256 hash according to the hashlib library's documentation.
- The computed hash does not contain any null bytes (NULs).
- The computed hash contains only hexadecimal digits (0-9, A-F, a-f).

COVERAGE

src/pytest_llm_report/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 for both computations.

Why Needed: This test prevents a potential bug where the hash computation and content hash are not consistent due to differences in the way Python's `hashlib` library handles file paths and byte arrays.

Key Assertions:

- The computed SHA-256 hash of the file should match its content hash.
- The path to the file should be a valid absolute or relative path.
- The file should not be empty.
- The file should have a valid UTF-8 encoding.
- The file should not contain any binary data other than the expected content.
- The file's mode (e.g., 'w', 'r', etc.) should be compatible with Python's `hashlib` library.
- The computed hash should be identical to the provided content hash even if the file is modified or deleted.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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 `compute_file_sha256` function correctly hashes a file.

Why Needed: This test prevents a potential bug where the hash is not generated correctly due to incorrect file contents or encoding.

Key Assertions:

- The length of the computed hash should be exactly 64 bytes.
- The hash should be generated from the correct file contents (i.e., 'hello world' in this case).
- The hash should be generated without any errors or exceptions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	5 lines (ranges: 44-48)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where two different keys could produce the same signature, which would be unexpected behavior for an HMAC-based system.

Key Assertions:

- Verify that the computed signature is different from the expected signature when using different keys.
- Check if the computed signature has a different value than the expected signature.
- Ensure the computed signature does not match the expected signature even if they are identical.
- Test if the difference in signatures is due to the actual key used for computation.
- Verify that the computed signature is unique when using different keys.
- Check if the expected signature matches the computed signature when using different keys.
- Ensure the computed signature has a different value than the expected signature even if they 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 61)

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

AI ASSESSMENT

Scenario: Verifies the production of an HMAC signature with a given key.

Why Needed: Prevents a potential issue where the HMAC length is not correctly calculated when using a secret key.

Key Assertions:

- The length of the generated HMAC should be exactly 64 bytes.
- The HMAC should be generated using the provided secret key.
- No other parameters (such as message and digest) should be used to generate the signature.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 61)

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

AI ASSESSMENT

Scenario: The function `compute_sha256` is called with the same input data (b'test') and produces the same output hash.

Why Needed: This test prevents a bug where different inputs to the `compute_sha256` function produce different hashes, potentially leading to inconsistent results in downstream applications.

Key Assertions:

- The two calls to `compute_sha256(b'...')` should return the same hash object.
- The two calls to `compute_sha256(b'...')` should return the same bytes object.
- The output hash of `compute_sha256(b'...')` should be equal to the input hash.
- The output hash of `compute_sha256(b'...')` should not change even if the input is modified (e.g., by appending a new character).
- The output hash of `compute_sha256(b'...')` should produce the same hash when run multiple times with the same input data.
- The function should raise an error or return a specific value when given invalid input data (e.g., non-string, non-byte string).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/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 length is not consistent across different test runs or environments.

Key Assertions:

- The length of the output should be exactly 64 characters.
- The hash should have 64 hexadecimal digits.
- No leading zeros are present in the hash.
- No trailing zeros are present in the hash.
- All hexadecimal digits are present and distinct.
- No duplicate hexadecimal digits are present.
- The hash 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` 70ms 3

AI ASSESSMENT

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

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

Key Assertions:

- The 'pytest' package should be present in the `get_dependency_snapshot()` output.
- The 'pytest' package should have been detected by the function.
- The presence of 'pytest' in the snapshot indicates that it is a required dependency.
- Without 'pytest', the test would fail due to missing dependencies.
- Including 'pytest' ensures compatibility with pytest testing framework.
- This test helps ensure that the `get_dependency_snapshot()` function works correctly for pytest packages.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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 72ms 3

AI ASSESSMENT

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

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

Key Assertions:

- snapshot is an instance of dict
- snapshot has no attributes other than __dict__
- snapshot does not contain any non-essential keys
- snapshot contains only essential package information
- snapshot has the correct number of packages
- snapshot includes all required dependencies
- snapshot is a dictionary with the expected structure

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 the `load_hmac_key` function with a loaded HMAC key.

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

Key Assertions:

- The function should successfully load the HMAC key from the specified file.
- The loaded HMAC key should match the expected value of 'my-secret-key'.
- The `load_hmac_key` function should not throw any exceptions when loading an HMAC key from a valid file.
- The `load_hmac_key` function should return the correct HMAC key for the given configuration.
- The `load_hmac_key` function should handle cases where the file is missing or cannot be read due to permissions issues.
- The `load_hmac_key` function should not throw any exceptions when loading an HMAC key from a non-existent file.
- The `load_hmac_key` function should correctly handle cases where the file path is incorrect or malformed.

COVERAGE

src/pytest_llm_report/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 that the function returns None when a missing key file is provided.

Why Needed: This test prevents a potential bug where the function would return a non-None value if a key file does not exist.

Key Assertions:

- The function should raise a ValueError or return None when a key file does not exist.
- The function should not attempt to load the HMAC key from the provided key file.
- The function should return an error message indicating that the key file is missing.
- The function should not throw any exceptions, but instead return a meaningful result.
- The function's behavior should be consistent across different Python versions and environments.
- The function's documentation should clearly indicate what happens when a 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 `load_hmac_key` function returns `None` when no key file is specified.

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

Key Assertions:

- The `load_hmac_key` function should return `None` if no key file is provided.
- No exception should be raised when no key file is specified.
- The function should correctly handle the absence of a key file configuration.
- The function should not throw an error or raise an exception when no key file is present.
- The function's behavior should align with the expected requirements for HMAC key loading.

COVERAGE

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

Why Needed: Prevents a regression where aggregation defaults are not set to sensible values, potentially leading to unexpected behavior or errors.

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: The test verifies that the capture failed output default setting is set to False.

Why Needed: This test prevents a bug where the default capture failed output setting would be enabled by default, potentially leading to unexpected behavior or errors.

Key Assertions:

- config.capture_failed_output
- assert config.capture_failed_output == False
- The capture failed output is not set to True.
- The capture failed output is not set to False by default.
- The capture failed output setting is not explicitly enabled in the configuration.
- The capture failed output setting is not disabled by default.
- The capture failed output setting is not properly validated or checked for in the test.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- config.llm_context_mode is equal to 'minimal'
- config.llm_context_mode is 'minimal'
- config.llm_context_mode is not 'default'
- config.llm_context_mode is not 'minimal'
- get_default_config() returns a configuration with llm_context_mode set to 'minimal'
- the value of config.llm_context_mode is not 'minimal'
- the value of config.llm_context_mode is '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 without a clear reason or context.

Key Assertions:

- config.is_llm_enabled() == False
- config.get_llm_enabled_value() == False
- get_default_config().llm_enabled() == False
- not config.is_llm_enabled() in [True, 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	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: Verifies that the `TestConfigDefaults` class has a default setting to omit tests from coverage.

Why Needed: This test prevents a regression where the `TestConfigDefaults` class does not correctly set the `omit_tests_from_coverage` flag by default.

Key Assertions:

- The `config.omit_tests_from_coverage` attribute is set to True.
- The `get_default_config()` function returns an instance of `TestConfigDefaults` with a `omit_tests_from_coverage` attribute set to True.
- The `TestConfigDefaults` class has a default setting for the `omit_tests_from_coverage` flag.
- The `TestConfigDefaults` class correctly sets the `omit_tests_from_coverage` flag by default.
- The `get_default_config()` function returns an instance of `TestConfigDefaults` with the correct default value for `omit_tests_from_coverage`.
- The `config.omit_tests_from_coverage` attribute is a boolean value indicating whether to omit tests from 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/options.py`	`3 lines (ranges: 107, 147, 233)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the provider is not set to 'none' in case of privacy requirements.

Key Assertions:

- The config object has a 'provider' attribute with value 'none'.
- The 'provider' attribute in the config object matches the expected default value when it's None.
- The provider setting does not override any other settings, ensuring consistent behavior.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The 'secret' keyword is present in the list of excluded globs.
- The '.env' file is not present in the list of excluded globs.
- No other keywords or patterns are present in the excluded globs.
- All secret files and directories should be excluded from the LLM context by default.
- Any sensitive information like secret files should be handled properly to prevent data exposure.
- The test ensures that only specific files and directories are excluded from the LLM context.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the output of the full pipeline is deterministic by comparing it to a sorted list.

Why Needed: This test prevents regression where the order of reported tests changes due to external factors.

Key Assertions:

- nodeids should be in ascending order
- nodeids should not contain duplicates
- nodeid 'z_test.py::test_z' is present in the sorted list
- nodeid 'a_test.py::test_a' is present in the sorted list
- nodeid 'm_test.py::test_m' is present 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 in cases where the test suite is empty, ensuring the report contains no invalid or incomplete information.

Key Assertions:

- The total count of tests in the report should be zero.
- The summary section of the report should contain an 'empty' key with a value of zero.
- There should be no missing or invalid test data in the report.

COVERAGE

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

PASSED `tests/test_integration_gate.py::TestFullPipeline::test_html_report_generation` 31ms 5

AI ASSESSMENT

Scenario: 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 HTML report should exist at the specified path.
- The HTML report contains the expected content.
- The 'test_pass' string is present in the HTML report 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/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` 54ms 7

AI ASSESSMENT

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

Why Needed: This test prevents regression where the JSON report is not generated correctly due to missing or incorrect configuration settings.

Key Assertions:

- The 'report_json' attribute of the config object should be set to the path of the generated report.
- The 'report_html' attribute of the config object should be set to the path of the generated report.
- The JSON output from the report writer should contain a 'schema_version' key with the correct value.
- The total count in the summary section should match the number of tests run.
- The passed count should match the number of tests that were successfully executed.
- The failed count should match the number of tests that encountered an error during execution.
- The skipped count should match the number of tests that were intentionally skipped by the test runner.

COVERAGE

src/pytest_llm_report/_git_info.py	2 lines (ranges: 2-3)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	78 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	133 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-

	198, 202, 211-218, 222-223, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-322, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

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

AI ASSESSMENT

Scenario: Test that the ReportRoot has required fields.

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

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that `RunMeta` has aggregation fields.

Why Needed: Prevents regression where the schema compatibility test fails due to missing aggregation fields.

Key Assertions:

- is_aggregated is present in the meta data
- run_count is present in the meta 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 'RunMeta has run status fields' verifies that the RunMeta object contains the required status fields.

Why Needed: This test prevents a potential regression where the RunMeta object is missing certain status fields, potentially causing issues with the integration gate.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Tests that the schema version is defined and matches a semver-like format.

Why Needed: This test prevents regression when using outdated or incompatible versions of the schema.

Key Assertions:

- The schema version should be defined.
- The schema version should match a semver-like format (e.g., '1.2.3').
- The schema version should contain at least one dot (.) character.
- The schema version should not start with a zero (0).
- The schema version should not end with a zero (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 the required fields.

Why Needed: This test prevents a bug where the `TestCaseResult` object is missing required fields, potentially causing unexpected behavior or errors in downstream processing.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the correct provider is not returned due to a mismatch in the expected class name.

Key Assertions:

- The method `__class__.__name__` of the `provider` object should return 'GeminiProvider'.
- The attribute `provider` should have an instance of `GeminiProvider` as its value.
- The correct provider should be returned for a valid configuration.
- An error message or exception should not be raised when calling `get_provider` with the expected provider.
- The test should fail if the incorrect class name is used instead of 'GeminiProvider'.
- A custom error message should not be printed by the `get_provider` function.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 the configuration is set to 'litellm'.

Why Needed: This test prevents a potential bug where the `get_provider` function does not return an instance of LiteLLMProvider when the configuration is set to 'litellm'.

Key Assertions:

- provider.__class__ == 'LiteLLMProvider'
- provider.model == 'gpt-3.5-turbo'
- provider.name == 'liteellm'

COVERAGE

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

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

AI ASSESSMENT

Scenario: test_get_provider_with_none_provider returns NoopProvider.

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

Key Assertions:

- provider should be None
- NoopProvider should be returned by get_provider() when config.provider='none'
- get_provider() should raise an exception when config.provider is 'none'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 using 'ollama' as a provider, OllamaProvider is returned.

Why Needed: This test prevents a potential bug where the correct provider type is not detected for 'ollama'.

Key Assertions:

- provider.__class__.__name__ should be equal to 'OllamaProvider'.
- The provider instance has an attribute named 'model' with value 'llama3.2'.
- The provider instance has a method named 'get_http_url' that returns the correct URL.
- The provider instance does not raise any exceptions when getting the model.
- The provider instance is of type OllamaProvider, not another class.
- The provider instance's get_http_url method returns the correct URL for 'ollama'.
- The provider instance has a method named 'get_model' that returns the correct model name.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 case: Unknown provider raises ValueError when trying to get a provider.

Why Needed: Prevents regression where unknown providers are used without raising an error.

Key Assertions:

- The function `get_provider` should raise a `ValueError` with the message 'unknown' when called with an unknown provider.
- The exception message should contain the string 'unknown'.
- The exception message should be case-insensitive (e.g., '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_llm.py::TestLlmProviderContract::test_noop_implements_interface` 1ms 5

AI ASSESSMENT

Scenario: Test that NoopProvider implements LlmProvider interface correctly.

Why Needed: Prevents a potential bug where the NoopProvider is not implemented as required by the LlmProvider contract.

Key Assertions:

- The provider should have all required methods.
- The provider should have 'annotate' method.
- The provider should have 'is_available' method.
- The provider should have 'get_model_name' method.
- The provider should have 'config' attribute.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the annotate method of NoopProvider returns an empty annotation when no node is provided.

Why Needed: This test prevents a regression where the annotate method fails to return an annotation for a non-existent node.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: The 'test_get_model_name_empty' test verifies that the 'NoopProvider' class returns an empty string when given no configuration.

Why Needed: This test prevents a potential bug where the model name is not returned correctly if no configuration is provided.

Key Assertions:

- assert provider.get_model_name() == ''
- assert provider.get_model_name() != 'noop' (to account for edge case)
- assert provider.get_model_name() != 'default' (to account for edge case)
- assert provider.get_model_name() != 'unknown' (to account for edge case)
- assert isinstance(provider.get_model_name(), str) (to ensure it's a string)
- assert len(provider.get_model_name()) == 0 (to verify the length is 0)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 accessible and returns True for is_available method.

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

Key Assertions:

- The `is_available()` method of the `NoopProvider` class should return `True`.
- The `is_available()` method of the `NoopProvider` class should always be called before attempting to use it.
- The `Config` instance passed to the `NoopProvider` constructor should not prevent the provider from being available.
- The `noop` function defined in the `NoopProvider` class should not return any value when called without arguments.
- The `is_available()` method of the `NoopProvider` class should be able to handle cases where it is not explicitly called before use.
- The `Config` instance passed to the `NoopProvider` constructor should have a valid configuration that allows the provider to function correctly.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test annotation summary is printed when annotations run.

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

Key Assertions:

- The function `get_provider` of `llm.annotator` returns a `FakeProvider` instance.
- The `provider` attribute of the `TestCaseResult` object is set to `FakeProvider` instance.
- The `captured.out` variable contains the expected string 'Annotated 1 test(s) via litellm'.
- The `get_provider` function is called with a valid configuration.
- The `provider` attribute of the `TestCaseResult` object is set to a valid provider.
- The `FakeProvider` instance is used in the annotation process.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	65 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

Scenario: Test that the progress report is generated for annotating tests with LLM.

Why Needed: This test prevents regression where the progress report is not generated correctly, potentially leading to incorrect reporting of LLM annotation status.

Key Assertions:

- The test case should be able to generate a correct progress report.
- The progress report should include the name of the test being annotated and its annotation ID.
- The progress report should indicate that the annotation is in progress.
- The progress report should not contain any missing or empty lines.
- The progress report should only contain messages related to LLM annotations.
- The progress report should not contain any other types of messages (e.g. test results, errors).
- The progress report should be generated for all tests annotated with LLM.
- The progress report should not be empty for the first test annotated with LLM.

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 by ensuring LLM annotations do not skip opt-out tests or exceed the maximum number of tests.

Key Assertions:

- The 'tests/test_a.py::test_a' node should be called when running LLM annotations with opt-out enabled.
- The first test case should have an annotation result without LLM optimisation.
- The second and third test cases should not have any annotation results.
- The number of LLM annotations generated should not exceed the maximum limit set by config.
- The 'tests/test_a.py::test_a' node should be called when running LLM annotations with a maximum of 1 tests.
- The 'tests/test_b.py::test_b' and 'tests/test_c.py::test_c' nodes should not have any annotation results.

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 LLM annotations respect the requests-per-minute rate limit.

Why Needed: This test prevents a potential regression where LLM annotations may not respect the rate limit, potentially leading to inaccurate results or performance issues.

Key Assertions:

- provider.calls should contain all test nodes with the expected outcome.
- sleep_calls should contain the expected sleep duration.
- The time.sleep function is called at least twice for each test node.
- Each test node has been annotated once within the rate limit.
- No more than 30 requests per minute are made to the LLM annotation service.
- The number of calls to get_provider is equal to the number of tests passed.
- The sleep duration between calls to time.sleep is at least 2 seconds for each test node.

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 the annotation function skips unavailable providers with a suitable message.

Why Needed: To prevent the annotation process from failing when an unavailable provider is detected, and provide a clear indication of what went wrong.

Key Assertions:

- The test verifies that the annotation function correctly skips annotation when an unavailable provider is present.
- The test verifies that the message provided by the annotation function is accurate and informative.
- The test verifies that the annotation process does not fail unexpectedly due to an unavailable provider.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Tests that annotations are cached between runs and that the annotation function calls a provider when needed.

Why Needed: This test prevents regression where the annotation function is called without a previous run, potentially causing issues with cache consistency.

Key Assertions:

- The `provider.calls` assertion checks if the `get_provider` method of the `FakeProvider` instance was called before and after calling `annotate_tests`.
- The `test.llm_annotation` assertion checks if it is not `None` after calling `annotate_tests`.
- The `test.llm_annotation.scenario` assertion checks that the scenario used by `test_llm_annotation` matches '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: The `test_required_fields` test verifies that the schema has a 'required' field.

Why Needed: This test prevents a potential bug where the schema is not properly validated without the required fields.

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: The test verifies that the AnnotationSchema can correctly parse a dictionary containing required keys.

Why Needed: This test prevents potential issues where the AnnotationSchema is not properly configured or validated against expected input data.

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: tests/test_llm_contract.py::TestAnnotationSchema::test_schema_handles_empty

Why Needed: This test prevents a potential bug where the annotation schema does not handle empty input correctly.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 handles partial inputs correctly.

Why Needed: This test prevents a potential bug where the AnnotationSchema does not handle partial input correctly, potentially leading to incorrect results or errors.

Key Assertions:

- The schema's scenario attribute is set to 'Partial only'.
- The schema's why_needed attribute is empty. This indicates that there are no specific requirements for handling partial inputs.
- The assertion checks if the schema's scenario and why_needed attributes match the expected values.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- assert 'scenario' in ANNOTATION_JSON_SCHEMA['properties']
- assert 'why_needed' in ANNOTATION_JSON_SCHEMA['properties']
- assert 'key_assertions' in ANNOTATION_JSON_SCHEMA['properties']
- assert isinstance(ANNOTATION_JSON_SCHEMA, dict)
- assert ANNOTATION_JSON_SCHEMA is not None
- assert type(ANNOTATION_JSON_SCHEMA) == dict
- assert len(ANNOTATION_JSON_SCHEMA) > 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_llm_contract.py::TestAnnotationSchema::test_schema_to_dict 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `AnnotationSchema` instance correctly serializes to a dictionary.

Why Needed: This test prevents regression by ensuring the correct serialization of the schema to a dictionary.

Key Assertions:

- assertion 1: The value of 'scenario' in the resulting dictionary matches the specified scenario.
- assertion 2: The value of 'why_needed' in the resulting dictionary matches the specified why_needed.
- assertion 3: The key 'key_assertions' is present in the resulting dictionary and contains the expected list of 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: Tests the factory method to return a NoopProvider when the provider is set to 'none'.

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

Key Assertions:

- The function `get_provider(config)` should return a NoopProvider instance for the given configuration.
- The `isinstance(provider, NoopProvider)` assertion should pass when the returned provider is indeed a NoopProvider.
- The test should fail when the provider is set to 'none' as expected.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The `test_noop_is_llm_provider` test verifies that the `NoopProvider` class correctly inherits from `LlmProvider`.

Why Needed: This test prevents a potential regression where the `NoopProvider` class is incorrectly implemented as an LLM provider instead of a NoOp provider.

Key Assertions:

- The `provider` variable should be an instance of `LlmProvider`.
- The `provider` variable should have a type hint of `LlmProvider`.
- The `provider` variable should not contain any LLM-related code.
- The `provider` variable should only contain NoOp-related code.
- The `provider` variable should be able to be instantiated with the correct configuration.
- The `NoopProvider` class should correctly inherit from `LlmProvider` without any issues.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/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 regression where the NoopProvider does not return an annotation for functions that are not annotated with @noop.

Key Assertions:

- The annotation returned by the NoopProvider should be empty.
- The scenario of this test should be an empty string.
- The why_needed message should indicate that the NoopProvider should return an annotation for functions without @noop annotations.
- The key_assertions should include a check for an empty annotation.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `annotate` method does not return an instance of `TestCaseResult`, potentially causing issues downstream in the testing pipeline.

Key Assertions:

- The result has a 'scenario' attribute
- The result has a 'why_needed' attribute
- The result has a 'key_assertions' 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	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 node gracefully.

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

Key Assertions:

- ...
- ...
- ...

COVERAGE

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

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

AI ASSESSMENT

Scenario: The `provider.annotate` method should handle a `None` context for the `code` field without raising an error or returning an empty value.

Why Needed: This test prevents potential issues where the `provider.annotate` method returns an incorrect result when given a `None` context for the `code` field, potentially causing downstream errors in the LLM contract.

Key Assertions:

- The `result` variable is not `None` after calling `provider.annotate(test, 'code', None)`.
- The `result` variable contains valid data (e.g., a dictionary or an object) for the `code` field.
- The `result` variable has the correct type (e.g., `dict`, `object`) for the `code` field.
- The `provider.annotate` method is called with the correct arguments (`test`, `'code'`, `None`).
- The `provider.annotate` method returns a value that can be used to update the test result correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/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: The test prevents a potential bug where the annotate method is missing for some providers.

Key Assertions:

- provider_name in ['none', 'ollama', 'litellm', 'gemini']
- hasattr(provider, 'annotate')
- callable(provider.annotate)
- for provider_name in ['none', 'ollama', 'litellm', 'gemini']: has_attr(provider, 'annotate')

COVERAGE

src/pytest_llm_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 test verifies that the `annotate` method handles large contexts correctly.

Why Needed: This test prevents a potential regression where the `annotate` method fails with an error when dealing with very large contexts.

Key Assertions:

- The context is annotated successfully without raising any exceptions.
- The annotation does not exceed the maximum allowed size.
- The `annotate` method returns an empty list for very large contexts.
- No errors are raised when annotating a very large context.
- The context is properly cleaned up after annotation.
- The `annotate` method handles very large contexts without significant performance impact.
- The test passes even with the most extreme values of the context dimension.

COVERAGE

src/pytest_llm_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 test verifies that the LiteLLMProvider annotates missing dependencies correctly.

Why Needed: This test prevents a potential bug where the provider does not report missing dependencies, potentially leading to silent failures or incorrect results.

Key Assertions:

- The annotation error message is correct and includes the name of the missing dependency.
- The annotation error message includes the correct installation command for the missing dependency.
- The annotation error message includes the full path to the missing dependency.
- The annotation error message does not include any misleading or incomplete information about the missing dependency.
- The annotation error message is consistent across different environments and test runs.
- The annotation error message is clear and easy to understand, even for users who are not familiar with pip or dependencies.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/gemini.py	12 lines (ranges: 134, 136-139, 141-142, 160-164)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the GeminiProvider annotates a missing token in the configuration.

Why Needed: To prevent a bug where a missing API token causes the provider to fail.

Key Assertions:

- The `GEMINI_API_TOKEN` environment variable is not set before calling `annotate()`
- The `GEMINI_API_TOKEN` environment variable is set but empty (i.e., an error) after calling `annotate()`
- The `GEMINI_API_TOKEN` environment variable is set and has a non-empty value after calling `annotate()`

COVERAGE

src/pytest_llm_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 and rate limits logic runs correctly.

Why Needed: Prevents regressions due to incorrect token usage or missing rate limit checks.

Key Assertions:

- The `annotate` method of the `GeminiProvider` instance records a token on the limiter.
- The `annotate` method of the `GeminiProvider` instance checks for and reports the total number of tokens used by the provider.
- The rate limits logic is executed without errors or exceptions.
- The number of tokens recorded matches the expected value (123) in the response metadata.
- The limiter's token usage list contains only one entry with a count of 123.
- The `annotate` method does not raise an exception 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	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: Verify that the LLM provider annotates retries on rate limits correctly.

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

Key Assertions:

- The provider should retry after a certain time (e.g. 1 minute) if the API call exceeds the rate limit.
- The provider should retry with a different set of parameters (e.g. query, model) to avoid rate limiting.
- The provider should not retry immediately after the rate limit is exceeded.
- The provider should retry within a certain time window (e.g. 5 minutes) if the API call exceeds the rate limit.
- The provider should retry with a different set of parameters (e.g. query, model) to avoid rate limiting and ensure consistent results.
- The provider should not retry immediately after the rate limit is exceeded and the previous retry failed.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	181 lines (ranges: 32-34, 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-222, 225-227, 233-234, 238-240, 242-243, 274-277, 280-283, 286, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330-333, 335-341, 343, 346, 348-350, 352-355, 360-366, 368-369, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428,

	430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression in the LLM provider's behavior, ensuring that it correctly handles high annotation loads without exceeding the daily limit.

Key Assertions:

- The `annotate` method calls `self._rotate_models_on_daily_limit()` after annotating a large number of models.
- The total number of annotations annotated does not exceed the daily limit (1000).
- The time taken to annotate a large number of models is within a reasonable range (less than 10 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	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_annotate_skips_on_daily_limit` test verifies that the annotation skips when exceeding daily limit.

Why Needed: This test prevents a regression where the annotation does not skip when exceeding the daily limit, potentially causing issues with downstream processing.

Key Assertions:

- The `annotation` attribute is set to `True` before exceeding the daily limit.
- The `annotation` attribute is reset to `False` after exceeding the daily limit.
- The `skips_on_daily_limit` attribute is not updated when exceeding the daily limit.
- The `annotation` attribute is only set to `True` if the `skips_on_daily_limit` attribute is `False` before exceeding the daily limit.
- Exceeding the daily limit does not cause the annotation to be skipped.
- The test passes even when the `skips_on_daily_limit` attribute is `True` but the `annotation` attribute is still set to `True` after exceeding the 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 caused by incorrect annotation of invalid 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 test verifies that the exhausted model recovers after 24 hours.

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

Key Assertions:

- The recovered model should have the same accuracy as before exhaustion.
- The recovered model should have the same number of parameters as before exhaustion.
- The recovered model's training time should be less than or equal to 24 hours.
- The recovered model's inference time should be within a reasonable range (e.g., < 10 seconds).
- The recovered model's memory usage should not increase significantly after 24 hours.
- The recovered model's performance metrics (e.g., F1 score, mean squared error) should remain stable or improve slightly after exhaustion.

COVERAGE

src/pytest_llm_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 test verifies that the `fetch_available_models` method returns an error when no models are available.

Why Needed: This test prevents a potential regression where the `fetch_available_models` method might return an error even when there are models available in the cache.

Key Assertions:

- assertRaises(SystemExit) with pytest.raises(SystemExit)
- assertRaises(SystemExit) with mock.patch.object(monkeypatch, 'raise')
- the returned value is a SystemExit exception
- the error message contains 'no models found'
- the error message does not contain any other information
- the error message does not indicate that the cache is exhausted

COVERAGE

src/pytest_llm_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 an interval of 5 seconds.

Why Needed: This test prevents a potential regression where the model list does not update after a short period of time.

Key Assertions:

- The `refresh_interval` attribute is set to 5000 (5 seconds) before each test call.
- The `refreshed_model_list` attribute is an empty list at the start of each test call.
- After calling `refresh_model_list()`, the `refreshed_model_list` attribute contains a non-empty list of models.
- The length of the `refreshed_model_list` attribute increases by 1 after each test call.
- The `refresh_interval` attribute is not set to 0 when the test function is called multiple times in quick succession.
- The `refresh_model_list()` method does not modify the original model list but returns a new one instead.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	169 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-222, 225-227, 274-277, 280-283, 286, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-341, 343, 346, 348-350, 352-355, 360-366, 368-369, 374-377, 381-382, 385-387, 391-393, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the LLM provider does not surface completion errors during annotation.

Key Assertions:

- The annotation should contain an error message indicating a completion error.
- The error message should be 'boom'.
- The annotation should have an error attribute that contains the string 'boom'.
- The annotation should indicate that the test case failed due to a completion error.
- The annotation should not return any value (i.e., it should be a CaseResult with outcome='passed').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: Prevents regression where the provider incorrectly accepts or ignores invalid key_assertions payloads.

Key Assertions:

- Invalid response: key_assertions must be a list
- Key assertion 'oops' is not a valid JSON object

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 an error when a required library is not installed.

Key Assertions:

- The annotation message includes the correct error message for installing the 'litellm' library.
- The annotation message does not include any misleading or incorrect information about the installation process.
- The annotation message provides a clear and concise way to inform users of the required dependency.
- The test case is able to pass successfully even if the 'litellm' library is not installed.
- The test case can be relied upon to provide accurate and helpful error messages for missing dependencies.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 LiteLLM provider annotates a successful response with the correct key assertions and confidence level.

Why Needed: This test prevents regressions by ensuring that the provider correctly annotates responses as 'status ok' and redirects to the expected URL.

Key Assertions:

- The annotation should contain the correct scenario.
- The annotation should contain the correct why needed message.
- The annotation should include the key assertions in its confidence level.
- The annotation should capture the model used by the provider.
- The annotation should capture the messages sent to the user during the login process.
- The annotation should include the expected URL redirection in its 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 'litellm' module.

Why Needed: Prevents a potential bug where the provider does not detect the 'litellm' module if it is not installed.

Key Assertions:

- The 'litellm' module should be present in sys.modules.
- The 'litellm' module should have been imported successfully.
- The 'litellm' module should be available as a provider for the LiteLLMProvider.
- The 'litellm' module should not cause any import errors when trying to instantiate it.
- The 'litellm' module should have its __name__ attribute set correctly.
- The 'litellm' module should have been registered in sys.modules with the correct key.
- The provider for 'litellm' should be available and functional.

COVERAGE

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

Key Assertions:

- It should not raise an exception when the context length is too large.
- It should return an error message indicating that the context length is too large.
- The returned error message should include information about the maximum allowed context length.
- The function should handle cases where the input is a list or tuple with more elements than the maximum allowed context length.
- The function should not raise an exception when the input is a string with more characters than the maximum allowed context length.
- The returned error message should be informative and easy to understand for users of the annotate function.
- The test should pass even if the input is a list or tuple that exceeds the maximum allowed context length by one element.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	33 lines (ranges: 52-53, 72, 75-76, 78, 165, 167-173, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/ollama.py	15 lines (ranges: 40-41, 47, 50, 52, 54-55, 57-60, 62-63, 66-67)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the Ollama provider correctly annotates a call to _call_ollama with an error message when it fails.

Why Needed: This test prevents regression in handling call errors, ensuring that the annotation is accurate and informative even when the underlying function raises an exception.

Key Assertions:

- The annotation should include the string 'Failed after 3 retries. Last error: boom'.
- The annotation should indicate that the call failed with a RuntimeError.
- The annotation should specify the exact error message 'boom' to avoid confusion with other potential errors.
- The annotation should not include any additional information about the underlying system prompt.
- The annotation should only include the error message, without any context or details about the call.
- The annotation should be consistent across all test cases that use the Ollama provider.
- The annotation should not interfere with other tests that may also annotate calls to _call_ollama with different 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/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 OllamaProvider should report an error when annotating a function without the required httpx dependency.

Why Needed: This test prevents a potential bug where the provider incorrectly reports missing dependencies, potentially leading to incorrect or misleading error messages.

Key Assertions:

- The annotation message should include the correct error message indicating that httpx is not installed.
- The annotation message should be specific about which dependency is required (httpx) and how to install it (pip install httpx).
- The test case should pass with an error message that indicates the missing dependency and provides a clear solution for installation.
- The provider's behavior should change when the test_case function is annotated, indicating that the annotation was successful despite the missing dependency.
- The test result should be marked as passed even if the annotation fails, to reflect the correct outcome of the test.
- The error message should not include any misleading information or assumptions about the user's environment.
- The provider should correctly report the version of httpx required for the test_case function.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 ensuring that the correct response is returned from the API.

Key Assertions:

- Check status to ensure the API returns a successful response
- Validate token to ensure it's properly authenticated
- Verify the response contains the expected JSON structure

COVERAGE

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

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

AI ASSESSMENT

Scenario: Ollama provider makes correct API call.

Why Needed: Prevents regression in Ollama provider's API call functionality.

Key Assertions:

- The response from the Ollama model is 'test response'.
- The URL of the API call is set to `http://localhost:11434/api/generate`.
- The model used by the Ollama provider is correctly identified as `llama3.2:1b`.
- The prompt used for the API call is set to `test prompt`.
- The system prompt used for the API call is set to `system prompt`.
- The stream parameter of the response is set to `False`.
- The timeout of the API call is correctly set to 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 the Ollama provider uses the default model when not specified.

Why Needed: This test prevents a regression where the default model is not used by the provider.

Key Assertions:

- The 'model' key in the captured response should be set to 'llama3.2'.
- The 'model' key in the captured response should contain only strings or None.
- The 'model' key in the captured response should not be empty.
- The default model 'llama3.2' is present in the captured response.
- The absence of a specified model is handled correctly and the default model is used.
- The provider's _call_ollama method does not raise an error when no model is provided.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 Ollama provider fails to return an error when the server is not running.

Key Assertions:

- The method _check_availability() of the OllamaProvider instance should raise a ConnectionError exception when the server is unavailable.
- The method _check_availability() of the OllamaProvider instance should set the 'is_available' attribute to False.
- The method _check_availability() of the OllamaProvider instance should not return any value (i.e., it should be a no-op).
- The method _check_availability() of the OllamaProvider instance should raise an exception when the server is unavailable.
- The 'is_available' attribute of the provider instance should be set to False after calling _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	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	6 lines (ranges: 87-88, 90-91, 93-94)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_check_availability_non_200` 1ms 5

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the provider incorrectly returns True for non-200 status codes, leading to incorrect usage of the API.

Key Assertions:

- The method _check_availability() is called on the OllamaProvider instance.
- The status code returned by the _check_availability() method is not 200.
- The provider's availability is set to False.
- The provider does not return True for non-200 status codes.
- The provider raises an exception when checking availability with a non-200 status code.
- The provider returns a response with a status code greater than 200.
- The provider sets the 'Content-Type' header of the response to 'application/json'.
- The provider does not set the 'Content-Type' header of the response to 'application/json' for non-200 status codes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 87-88, 90-92)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the Ollama provider checks availability via /api/tags endpoint and returns True when successful.

Why Needed: This test prevents a potential bug where the provider does not check for availability before returning, potentially leading to unexpected behavior or errors.

Key Assertions:

- The '/api/tags' URL is present in the provided URL.
- The response status code is 200.
- The provider returns True when checking availability.
- The provider's _check_availability method is called with no arguments.
- The provider does not raise an exception if it cannot connect to the Ollama host.
- The provider does not log any errors or warnings.
- The provider's configuration is valid and matches the expected values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 87-88, 90-92)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the provider might incorrectly return `False` for local configurations.

Key Assertions:

- provider.is_local() == True
- provider.config.provider == 'ollama'
- provider.config is not None
- is_local() is True
- not is_local() 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/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 parsing a response with invalid JSON.

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

Key Assertions:

- 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 Ollama provider rejects invalid key_assertions payloads in its _parse_response method.

Why Needed: This test prevents regression where the Ollama provider incorrectly accepts or ignores invalid key_assertions payloads.

Key Assertions:

- the response data should be a dictionary with a 'key_assertions' key
- the 'key_assertions' value should be a list of strings
- the error message should indicate that the 'key_assertions' field is required and must be a list
- the provider should reject any invalid or missing 'key_assertions' payloads
- the provider should raise an exception when encountering an invalid 'key_assertions' payload

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The extracted JSON is a valid JSON object.
- The JSON object contains only strings and numbers.
- The JSON object does not contain any arrays or objects with nested keys.
- The JSON object has the correct indentation (4 spaces).
- No null values are present in the JSON object.
- The JSON object does not contain any undefined variables or functions.
- All string values in the JSON object are enclosed in double quotes.
- The JSON object is a valid JSON 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	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	6 lines (ranges: 38, 42-44, 46-47)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The provided test verifies that the Ollama provider can extract JSON from a plain markdown fence without any language specified.

Why Needed: This test prevents regression in case the provider encounters an issue when parsing responses with no language specified.

Key Assertions:

- The response is not empty.
- The response starts with `"`.
- The response ends with `"""`.
- Each line of the response contains only two characters (either `"` or `"""`).
- There are no special markdown syntaxes in the response.
- The provider correctly extracts JSON from the response without any language specified.
- The extracted JSON is a valid Python dictionary.
- The extracted JSON does not contain any invalid characters.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 that the Ollama provider correctly parses valid JSON responses.

Why Needed: Prevents bugs in the LLM providers by ensuring they return expected results for successful 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 the `CoverageEntry` class correctly serializes its attributes.

Why Needed: This test prevents a potential bug where the serialized data does not match the expected format.

Key Assertions:

- The `file_path` attribute is set to 'src/foo.py'.
- The `line_ranges` attribute is set to '1-3, 5, 10-15'.
- The `line_count` attribute is set to 10.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`4 lines (ranges: 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 the test collection.

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

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.
- All required keys ('file_path', 'line_ranges', and 'line_count') are present in the dictionary.
- The values of the 'file_path', 'line_ranges', and 'line_count' keys are correct and follow the expected format.

COVERAGE

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

Why Needed: To prevent a bug where the coverage entry's line ranges are not correctly serialized.

Key Assertions:

- The 'file_path' key should be present and contain the expected value.
- The 'line_ranges' key should be present and contain the correct string representation.
- The 'line_count' key should be present and contain the expected value.
- The line ranges in the 'line_ranges' key should be correctly formatted (e.g., '1-3, 5, 10-15')
- Each range in the 'line_ranges' key should be a valid Python range object (e.g., [1, 3], [5, 7])
- The line count should be an integer value (0 or positive)
- If the file path is empty, the 'file_path' key should still contain an empty string.
- If the line ranges are not correctly formatted, they should raise a ValueError.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where an empty annotation would cause the model to fail or produce incorrect results without any specified parameters.

Key Assertions:

- annotation.scenario == "" (empty string)
- annotation.why_needed == "" (empty string) (to prevent model failure)
- annotation.key_assertions == [] (to ensure no specific keys are set)
- assert annotation.confidence is None (to prevent incorrect confidence values)
- assert annotation.error is None (to prevent incorrect 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestLlmAnnotation::test_to_dict_minimal` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `LlmAnnotation` object's `to_dict()` method returns a dictionary with required fields.

Why Needed: This test prevents regression by ensuring that the minimal annotation is correctly serialized without any optional fields.

Key Assertions:

- The 'scenario' key should be present in the dictionary.
- The 'why_needed' key should be present in the dictionary.
- The 'key_assertions' key should be present in the dictionary.
- The 'confidence' field should not be included in the dictionary when it is None.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	8 lines (ranges: 104-107, 109, 111, 113, 115)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestLlmAnnotation::test_to_dict_with_all_fields` 1ms 3

AI ASSESSMENT

Scenario: Test that the full annotation is included in the dictionary.

Why Needed: Prevents a potential bug where only partial fields are included in the dictionary.

Key Assertions:

- The 'scenario' field should be present and have the correct value.
- The 'confidence' field should have a value greater than or equal to 0.95.
- The 'context_summary' field should contain the expected keys ('mode' and 'bytes') with correct values.
- All other required fields (error, token_present) should be present and have correct values.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test Default Report

Why Needed: Prevents a potential bug where the report does not include required schema version and collection error information.

Key Assertions:

- The 'schema_version' key should be present in the dictionary with value equal to SCHEMA_VERSION.
- The 'tests' key should be an empty list.
- The 'warnings' key should not be included in the dictionary (excluding the empty lists).
- The 'collection_errors' key should also not be included in the dictionary (excluding the empty lists).

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test Report with Collection Errors should be verified by checking the presence of collection errors in the report.

Why Needed: This test prevents a regression where a report might not include all collection errors, potentially leading to incorrect reporting or missed issues.

Key Assertions:

- The 'collection_errors' key is present in the report dictionary and contains exactly one error.
- The value of 'nodeid' for the first error is set to 'test_bad.py'.
- All collection errors are included in the report.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	58 lines (ranges: 207-209, 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508-510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test verifies that a ReportRoot instance with warnings is properly reported.

Why Needed: This test prevents a regression where the report does not include all warnings.

Key Assertions:

- The length of `d['warnings']` should be equal to 1.
- The value of `d['warnings'][0]['code']` should be 'W001'.
- All warnings in the report should have a non-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/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 by their nodeids.

Key Assertions:

- The list of nodeids returned by `to_dict()` matches the expected ordering.
- Each nodeid appears once in the list.
- All nodeids appear before any other test results.
- Nodeids are present in the order they were defined (a_test.py::test_a, m_test.py::test_m, z_test.py::test_z).
- The nodeids are sorted alphabetically (z_test.py::test_z comes first).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	71 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestReportWarning::test_to_dict_with_detail 1ms 3

AI ASSESSMENT

Scenario: Test `test_to_dict_with_detail` verifies that a `ReportWarning` instance is converted to a dictionary with the correct 'detail' key.

Why Needed: This test prevents a potential issue where the 'detail' field in the warning dictionary is not correctly populated for certain types of warnings.

Key Assertions:

- The value of the 'detail' key in the warning dictionary should be '/path/to/file'.
- The 'detail' key should contain the path to the file that triggered the warning.
- The 'detail' field should not be missing from the warning dictionary.
- The 'detail' field should have a non-empty value for warnings with a code of 'W001'.
- The 'detail' field should include the message and code of the warning. In this case, it includes only the message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: The test verifies that the `to_dict()` method of `ReportWarning` returns a dictionary with only 'code' and 'message' keys.

Why Needed: This test prevents a warning from being reported when a `ReportWarning` object is created without specifying any details.

Key Assertions:

- The expected output should be a dictionary with 'code' and 'message' keys.
- The 'detail' key should not be present in the dictionary.
- The value of 'code' should be exactly 'W001'.
- The value of 'message' should be exactly '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	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: Prevents regression where RunMeta is missing necessary aggregation fields.

Key Assertions:

- The 'run_id' key should contain the value 'run-123'.
- The 'run_group_id' key should contain the value 'group-456'.
- The 'is_aggregated' key should be True.
- The 'aggregation_policy' key should be set to 'merge'.
- The 'run_count' key should have a value of 3.
- The length 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: Tests that LLM fields are excluded when annotations are disabled.

Why Needed: This test prevents a regression where the LLM fields are included even when annotations are disabled.

Key Assertions:

- The 'llm_annotations_enabled' key is not present in the data.
- The 'llm_provider' key is not present in the data.
- The 'llm_model' key is not present in the data.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that LLM traceability fields are included when enabled.

Why Needed: Prevent regression in model tracing functionality by ensuring all required fields are present.

Key Assertions:

- data['llm_annotations_enabled'] is True
- data['llm_provider'] == 'ollama'
- data['llm_model'] == 'llama3.2:1b'
- data['llm_context_mode'] == 'complete'
- data['llm_annotations_count'] == 10
- data['llm_annotations_errors'] == 2

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	40 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407-419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 are included in non-aggregated runs, which can be misleading and cause confusion.

Key Assertions:

- The 'source_reports' key is present in the dictionary.
- The value of 'is_aggregated' is set to False.
- The presence of 'source_reports' in the dictionary indicates an aggregated report.
- A non-aggregated run should not include source reports.
- The absence of 'source_reports' in the dictionary suggests that a non-aggregated run does not contain source reports.
- Including source reports in a non-aggregated run can be misleading and cause confusion.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test RunMeta to dict with all optional fields.

Why Needed: Prevents regression in case of missing or outdated plugin version, which could lead to incorrect aggregation policy.

Key Assertions:

- The 'git_sha' field is set to 'abc1234'.
- The 'git_dirty' field is True.
- The 'repo_version' field is set to '1.0.0'.
- The 'repo_git_sha' field is set to 'abc1234'.
- The 'repo_git_dirty' field is True.
- The 'plugin_git_sha' field is set to 'def5678'.
- The 'plugin_git_dirty' field is False.
- The 'config_hash' field is set to 'def5678'.
- The length of the source reports list is 1.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	49 lines (ranges: 277-279, 281-283, 364-380, 382-405, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test RunMeta includes run status fields.

Why Needed: The test prevents a potential bug where the 'RunMeta' object is missing certain required fields for accurate representation of its run status.

Key Assertions:

- assert d['exit_code'] == 1
- assert d['interrupted'] is True
- assert d['collect_only'] is True
- assert d['collected_count'] == 10
- assert d['selected_count'] == 8
- assert d['deselected_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/models.py	29 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: Prevents regression where the schema version is not in a valid semver format.

Key Assertions:

- The schema version should be split into three parts (major, minor, patch).
- Each part of the schema version should be a digit.
- All digits in the schema version should be non-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)

PASSED `tests/test_models.py::TestSchemaVersion::test_schema_version_in_report_root` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `ReportRoot` class includes the correct `schema_version`.

Why Needed: This test prevents a potential bug where the schema version is not included in the report root.

Key Assertions:

- The `report.schema_version` attribute should be set to `SCHEMA_VERSION`.
- The `to_dict()` method of `ReportRoot` should return an object with a `schema_version` key set to `SCHEMA_VERSION`.
- The value of the `schema_version` key in the `to_dict()` output should match `SCHEMA_VERSION`.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test coverage entry serialization.

Why Needed: CoverageEntry should be able to serialize correctly without any errors.

Key Assertions:

- The 'file_path' key is present and contains the expected value.
- The 'line_ranges' key is present and contains the expected values.
- The 'line_count' key is present and contains the expected value.
- Each assertion passes, indicating coverage entry serialization 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/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 by ensuring that the minimal annotation format includes all necessary information.

Key Assertions:

- The 'scenario' key should be present in the dictionary.
- The 'why_needed' key should be present in the dictionary.
- The 'key_assertions' key should be present in the dictionary.
- The 'confidence' key should not be present in the dictionary when it is None.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the `to_dict()` method of `SourceReport` with a `run_id`.

Why Needed: Prevents regression where a `SourceReport` instance is created without a `run_id`.

Key Assertions:

- The `run_id` key in the resulting dictionary should be 'run-1'.
- The value of the `run_id` key should match the provided `run_id`.
- If no `run_id` is provided, the `run_id` key should still be present but empty ('').
- If an invalid `run_id` (e.g., not a string) is provided, it should raise an error.
- The `to_dict()` method should return a dictionary with the required keys (`run_id`) and value.
- The `to_dict()` method should handle cases where the `SourceReport` instance has no `run_id` or an empty string for `run_id`.
- If the `run_id` is not present in the resulting dictionary, it should be ignored.

COVERAGE

src/pytest_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: Tests coverage-related functionality.

Why Needed: This test prevents a bug where the `CoverageEntry` class does not correctly serialize to JSON.

Key Assertions:

- The 'file_path' key in the dictionary should be equal to the expected value.
- The 'line_ranges' key in the dictionary should be equal to the expected value.
- The 'line_count' key in the dictionary should be equal to the expected value.
- The 'CoverageEntry' class correctly serializes to JSON when creating a `CoverageEntry` object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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 in cases where a minimal result is expected but not provided.

Key Assertions:

- The 'nodeid' field of the minimal result should be set to 'test_foo.py::test_bar'.
- The 'outcome' field of the minimal result should be set to 'passed'.
- The 'duration' field of the minimal result should be set to 0.0.
- The 'phase' field of the minimal result 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` includes a coverage list.

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

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 'coverage' entry should be 'src/foo.py'.
- The file path is not empty or None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 Result with LLM Opt-Out should include flag.

Why Needed: To prevent regression in cases where the user explicitly chooses to opt-out of using the Large Language Model (LLM) for result calculation.

Key Assertions:

- The value of llm_opt_out is set to True in the result dictionary.
- The key 'llm_opt_out' exists in the result dictionary and its value matches the expected boolean value.
- The test verifies that the LLMMergeResult object correctly includes a flag indicating opt-out from using LLM for result calculation.

COVERAGE

src/pytest_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 `TestCaseResult` object includes rerun fields.

Why Needed: This test prevents a regression where the `rerun_count` and `final_outcome` are not correctly populated in the `TestCaseResult` object when rerunning tests.

Key Assertions:

- The value of `rerun_count` is set to 2.
- The value of `final_outcome` is set to 'passed'.
- The `rerun_count` field should be present and have a value of 2.
- The `final_outcome` field should be present and have a value of 'passed'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	19 lines (ranges: 161-165, 167, 169, 171, 173-176, 178, 180, 182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_models.py::TestTestCaseResult::test_result_without_rerun_excludes_fields` 1ms 3

AI ASSESSMENT

Scenario: Test case "test_result_without_rerun_excludes_fields" verifies that the `result` dictionary does not contain 'rerun_count' and 'final_outcome' keys.

Why Needed: This test prevents regression where a test might incorrectly assume that reruns do not affect the result.

Key Assertions:

- The `result` dictionary should not contain 'rerun_count' key.
- The `result` dictionary should not contain 'final_outcome' key.
- The `result` dictionary should only contain necessary fields (e.g. nodeid, outcome).
- If reruns are performed, the `result` dictionary should still be valid and include all necessary fields.
- If a test is rerun, it should not change the result of that run.
- Rerunning a test should not affect the final outcome or other non-rerun-related data in the `result` dictionary.
- The presence of 'rerun_count' or 'final_outcome' keys in the `result` dictionary should be checked for each test individually.

COVERAGE

`src/pytest_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 regression in the case where default values are not properly initialized.

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'
- not cfg.is_llm_enabled() 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, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function returns an instance of `Config`.
- The `provider` attribute of the returned `Config` object is set to `'none'`.
- A valid configuration with a different provider (e.g., 'some_provider') is not 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	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 that the `is_llm_enabled` check returns the correct result for different providers.

Why Needed: This test prevents a potential regression where the `is_llm_enabled` check is incorrectly returning True when it should be False for some providers.

Key Assertions:

- The `is_llm_enabled` method of the `Config` class returns `False` when set to `none` and `True` when set to `ollama`.
- When the `provider` is set to `none`, the `is_llm_enabled` method should return `False`.
- The `is_llm_enabled` method of the `Config` class returns `False` when the `provider` is not set.
- When the `provider` is set to `ollama`, the `is_llm_enabled` method should return `True`.
- The `is_llm_enabled` method of the `Config` class should return `True` when the `provider` is set to `ollama` and it's an instance of `Config`.
- When the `provider` is set to `ollama`, the `is_llm_enabled` method should return `True`.
- The `is_llm_enabled` method of the `Config` class should not throw any exceptions when the `provider` is not set.

COVERAGE

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

PASSED `tests/test_options.py::TestConfig::test_validate_invalid_aggregate_policy` 1ms 3

AI ASSESSMENT

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-197, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options.py::TestConfig::test_validate_invalid_context_mode` 1ms 3

AI ASSESSMENT

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-189, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the `validate()` method of a `TestConfig` object returns an error message when an invalid include phase is specified.

Why Needed: This test prevents a potential bug where an invalid include phase is not properly validated and causes errors in the application.

Key Assertions:

- The `validate()` method should return exactly one error message.
- The error message should contain the string 'Invalid include_phase 'lunch_break'.
- The error message should be present in the first element of the list returned by `validate()`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-205, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test validation with an invalid provider.

Why Needed: Prevents a potential bug where the test fails due to an unexpected error message when validating an invalid provider.

Key Assertions:

- The configuration object should be validated successfully without any errors.
- The 'Invalid provider' error message should be present in the validation result.
- The error message should contain the exact phrase 'Invalid provider'.
- The test should fail if the provider is not recognized or is invalid.
- A single error message should be returned when validating an invalid provider.
- The configuration object should not throw any exceptions during validation.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test validation of numeric constraints for Config object.

Why Needed: This test prevents a potential bug where the llm_context_bytes is set to a value less than 1000, which could lead to unexpected behavior or errors in the LLM.

Key Assertions:

- cfg.validate() should return at least 5 error messages.
- The 'llm_context_bytes' constraint must be met (>= 1000).
- The 'llm_max_tests' constraint must be either 0, positive or no limit.
- The 'llm_requests_per_minute' constraint must be at least 1.
- The 'llm_timeout_seconds' constraint must be at least 1.
- The 'llm_max_retries' constraint must be either 0 or positive.
- All constraints (context_bytes, max_tests, requests_per_minute, timeout_seconds, max_retries) must be met.

COVERAGE

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

Why Needed: A valid configuration is required to prevent potential issues and ensure correct behavior.

Key Assertions:

- The `validate()` method of the `Config` class returns an empty list when a valid configuration is provided.
- No errors are reported when a valid configuration is passed to the `validate()` method.
- The `validate()` method does not throw any exceptions or raise an error when a valid configuration is used.
- A successful validation outcome is achieved without any issues or warnings.
- The `Config` class correctly handles a valid input and returns no 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 potential bug where aggregation options are not loaded correctly due to incorrect directory or policy settings.

Key Assertions:

- The aggregate_dir attribute is set to 'aggr_dir'.
- The aggregate_policy attribute is set to 'merge'.
- The aggregate_run_id attribute is set to 'run-123'.
- The aggregate_group_id attribute 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 handling of invalid integer values in INI.

Why Needed: Prevents a potential crash due to an invalid integer value in the INI file.

Key Assertions:

- The function `load_config` should not crash when encountering an invalid integer value in the INI file.
- The default value of `llm_max_retries` is 3, which is the expected behavior when a valid integer value cannot be found.
- The `getini` method's side effect does not cause any crashes or unexpected behavior for valid values.
- The test ensures that the function behaves correctly even if an invalid value is present in the INI file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	28 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263-267, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the `llm_coverage_source` option is correctly set to 'cov_dir' when loaded.

Why Needed: This test prevents a potential bug where the coverage source is not properly set, potentially leading to incorrect output or errors in the analysis.

Key Assertions:

- The value of `cfg.llm_coverage_source` should be equal to 'cov_dir'.
- The `llm_coverage_source` option should have been set to 'cov_dir' during configuration.
- The coverage source path should match the expected value '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: Verify that the default provider and report HTML are correctly loaded when no options are provided.

Why Needed: Prevents a potential bug where the test fails to load configuration due to missing provider or report HTML settings.

Key Assertions:

- The `provider` attribute of the configuration object is set to 'none'.
- The `report_html` attribute of the configuration object is set to `None`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 potential bug where the CLI overrides ini settings, potentially causing unexpected behavior or incorrect configuration.

Key Assertions:

- mock_pytest_config.getini.side_effect should be set to lambda key: 'ini_value' if key == 'llm_report_html' else None
- mock_pytest_config.option.llm_report_html should be set to 'cli_report.html'
- mock_pytest_config.option.llm_requests_per_minute should not be set in ini, but instead be 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 in the command line.

Key Assertions:

- The value of `llm_max_retries` should be set to 9.
- The value of `llm_max_retries` should match the expected value (in this case, 9).
- The configuration object should have a valid `llm_max_retries` attribute.
- The `llm_max_retries` option should not be changed unexpectedly after setting it in the command line.
- The test should fail if the `llm_max_retries` option is set to an invalid value (e.g. 0).
- The test should pass if the `llm_max_retries` option is correctly updated to 9.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	25 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282-283, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test loading values from ini options.

Why Needed: Prevents a potential bug where the test fails if the `getini` method is not called with a valid key.

Key Assertions:

- The value of `llm_report_provider` should be 'ollama'.
- The value of `llm_report_model` should be 'llama3'.
- The value of `llm_report_context_mode` should be 'balanced'.
- The value of `llm_report_requests_per_minute` should be 10.
- The value of `llm_report_max_retries` should be 5.
- The value of `llm_report_html` should be 'report.html'.
- The value of `llm_report_json` should be 'report.json'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	32 lines (ranges: 107, 147, 248, 251-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Tests the aggregation settings configuration.

Why Needed: Prevents a potential bug where the aggregate directory, policy, and include history are not correctly set for aggregated data.

Key Assertions:

- The `aggregate_dir` attribute is set to `/reports` as expected.
- The `aggregate_policy` attribute is set to `merge` as expected.
- The `aggregate_include_history` attribute is set to `True` as expected.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test Config with all output paths.

Why Needed: Prevents a regression where the test fails to verify that all required output files are present.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `capture_failed_output` attribute of the `Config` object is set to `True`.

Why Needed: This test prevents a potential issue where the capture settings are not properly configured, potentially leading to incorrect output or errors.

Key Assertions:

- config.capture_failed_output is True
- config.capture_output_max_chars is 8000

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `metadata_file` and `hmac_key_file` are correctly set in the test configuration.

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

Key Assertions:

- The value of `config.metadata_file` is correct and matches 'metadata.json'.
- The value of `config.hmac_key_file` is correct and matches '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: Tests the configuration of coverage settings.

Why Needed: Prevents a potential bug where coverage settings are not properly configured, potentially leading to incorrect test results or missed tests.

Key Assertions:

- config.omit_tests_from_coverage is set to False
- config.include_phase is set to "all"

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the specified custom exclude globs are included in the configuration.

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

Key Assertions:

- The string `*.pyc` is present in the `llm_context_exclude_globs` list.
- The string `*.log` is present in the `llm_context_exclude_globs` list.
- Custom exclude globs are included in the configuration.
- The custom exclude globs match the expected values.
- No custom exclude globs are excluded from the LLM context.
- The `llm_context_exclude_globs` list is updated correctly with custom exclude globs.
- Custom exclude globs are properly propagated to the LLM context.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `include_globs` configuration option includes only `.py` files.

Why Needed: Prevents a potential bug where include globs are not correctly applied to LLM context files.

Key Assertions:

- The `*.py` glob matches only files with the `.py` extension.
- The `*.pyi` glob matches only files with the `.pyi` extension.
- The `include_globs` configuration option is set correctly for LLM context files.
- The `llm_context_include_globs` attribute of the `Config` object contains the correct include globs.
- The `include_globs` attribute does not contain any invalid or missing glob patterns.
- The `include_globs` attribute does not contain any duplicate glob patterns.
- The `include_globs` attribute is set correctly for all supported file extensions (`.py`, `.pyi`).

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `include_pytest_invocation` attribute of the `Config` object is set to `False`.

Why Needed: This test prevents a regression where the invocation settings are not correctly configured for Pytest.

Key Assertions:

- The `include_pytest_invocation` attribute of the `config` object is set to `False`.
- The `include_pytest_invocation` attribute of the `config` object does not match its current value.
- The `include_pytest_invocation` attribute of the `config` object is not correctly set for Pytest invocation settings.
- The `config` object has an incorrect `include_pytest_invocation` attribute value.
- The `config` object's `include_pytest_invocation` attribute does not match its expected value.
- The `config` object's `include_pytest_invocation` attribute is set to a valid value for Pytest invocation settings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: Test the configuration of LLM execution settings.

Why Needed: Prevents regression in LLM execution settings configuration.

Key Assertions:

- The value of llm_max_tests is set to 50.
- The value of llm_max_concurrency is set to 8.
- The value of llm_requests_per_minute is set to 12.
- The value of llm_timeout_seconds is set to 60 seconds.
- The value of llm_cache_ttl_seconds is set to 3600 seconds (1 hour).
- The directory for the LLM cache is set to .cache.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the configuration of LLM parameter settings.

Why Needed: This test prevents a potential bug where the maximum characters for LLM parameters are not being set correctly.

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: Test the configuration of LLM settings for OLLAMA provider.

Why Needed: Prevents a potential bug where the model is not set correctly or the context bytes are too large, potentially causing issues with the LLM.

Key Assertions:

- The `provider` attribute in the test_llm_settings method should be equal to 'ollama'.
- The `model` attribute in the test_llm_settings method should be equal to 'llama3.2'.
- The value of `llm_context_bytes` in the test_llm_settings method should be 64000.
- The value of `llm_context_file_limit` in the test_llm_settings method should not exceed 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 is correctly set to `/project`.

Why Needed: This test prevents a potential issue where the repository root path is not properly configured, potentially leading to errors or unexpected behavior in subsequent tests.

Key Assertions:

- config.repo_root
- is_path('/project')
- is_absolute('/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 all valid include_phase values pass validation.

Why Needed: Prevents a potential bug where invalid or missing include_phase values are passed through to the validation process.

Key Assertions:

- The `validate()` method of the `Config` class does not return any errors for valid include_phase values.
- All include_phase values ('run', 'setup', 'teardown', and 'all') are present in the configuration.
- No error messages or warnings are raised when validating a valid include_phase 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	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: Verifies that the default exclude globs are correctly set.

Why Needed: This test prevents a potential bug where the default exclude globs do not include all necessary files.

Key Assertions:

- The function `llm_context_exclude_globs` is called with an empty list of arguments, and it should return a list containing only `.pyc`, `__pycache__/*`, and `*secret*`.
- The function `llm_context_exclude_globs` is called with the correct list of exclude globs, including all necessary files.
- If any of the required files are excluded from the default globs, an assertion error should be raised.

COVERAGE

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

PASSED tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_redact_patterns 1ms 3

AI ASSESSMENT

Scenario: Test default redact patterns with various configuration options.

Why Needed: Prevents regression when using the default configuration, ensuring that sensitive information is not exposed.

Key Assertions:

- The `--password` and `--token` flags are included in the default redact patterns.
- The pattern for `--api[_-]?key` includes a hyphenated key, which is expected to be redacted.

COVERAGE

src/pytest_llm_report/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 potential regression where the default provider and llm context mode are not set correctly.

Key Assertions:

- The config.provider should be 'none'.
- The config.llm_context_mode should be 'minimal'.
- The 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: Test that the `is_llm_enabled` method returns correct enabled status for different providers.

Why Needed: This test prevents regression in case a provider is changed or removed, and LLMs are not enabled by default.

Key Assertions:

- The method should return False when the provider is 'none'.
- The method should return True when the provider is 'ollama' (LLM).
- The method should return True when the provider is 'litellm' (LLM).
- The method should return True when the provider is '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

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-197, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_context_mode` 1ms 3

AI ASSESSMENT

Scenario: Test the validation of an invalid context mode.

Why Needed: Prevents a potential bug where an invalid context mode is passed to the llm, potentially causing unexpected behavior or errors.

Key Assertions:

- The function `Config(llm_context_mode='invalid')` should be called with a valid 'context_mode' string.
- An error message indicating that the provided 'llm_context_mode' is invalid should be returned by the `validate()` method.
- A specific error message containing the word 'Invalid' should be present in the error messages returned by the `validate()` method for 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 function `Config(include_phase='invalid')` should return an error.
- The error message for 'Invalid include_phase 'invalid'' should be present in the list of errors.
- The first error in the list should contain the string 'Invalid include_phase 'invalid'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-205, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_provider` 1ms 3

AI ASSESSMENT

Scenario: Test validates an invalid provider.

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

Key Assertions:

- The function `Config` is called with an invalid provider.
- The `validate()` method of the `Config` object returns exactly one error message.
- The error message contains the string 'Invalid provider 'invalid'.
- The test asserts that there is only one error in the list of errors returned by `config.validate()`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_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 invalid numeric values from being passed to the LLM.

Key Assertions:

- The function should return an error for llm_context_bytes = 999.
- The function should return an error for llm_max_tests = -1.
- The function should return an error for llm_requests_per_minute = 0.
- The function should return an error for llm_timeout_seconds = 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	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 empty configuration object is returned when the input config is valid.

Why Needed: Prevents a potential bug where an invalid config would cause the validation to fail with an empty list.

Key Assertions:

- The validate() method of the Config class should return an empty list for a valid config.
- An empty list should be returned when calling config.validate().
- A ValueError or other exception should not be raised when calling config.validate() for a valid config.
- The validate() method should only raise exceptions for invalid configurations, not for valid ones.
- The validate() method should return an empty list for all valid configurations.
- The validate() method should not throw any unexpected behavior for valid configurations.
- A TypeError or other exception should not be raised when calling config.validate() for a valid config.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 config defaults to safe values when no options are provided.

Why Needed: Prevents a potential bug where the config is set to an invalid or unexpected value.

Key Assertions:

- The `cfg` variable should be an instance of `Config`.
- The `cfg` variable should not have any registered options.
- The `cfg` variable's values should match the expected safe defaults.
- No additional configuration options should be present in the config.
- The `cfg` variable's type should be consistent with the `Config` class.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	33 lines (ranges: 107, 147, 248, 251-259, 261, 263-265, 270, 272-276, 278, 280, 282, 286, 288, 290-292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_plugin_integration.py::TestPluginConfigLoading::test_markers_exist_in_config` 1ms 2

AI ASSESSMENT

Scenario: Verify that the `pytestconfig` object is accessible in the test environment.

Why Needed: This test ensures that the plugin configuration is properly loaded and available for use during testing.

Key Assertions:

- The `pytestconfig` object should not be None.
- The `pytestconfig` object should have attributes (e.g., 'markers', 'plugins') accessible.
- The `pytestconfig` object should contain the expected configuration markers and plugins.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 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 in the LLM integration.

Why Needed: This test prevents a bug where the context marker causes an error, potentially leading to incorrect results or failures.

Key Assertions:

- assert True
- assert len(self.collection_errors) == 0
- assert all(isinstance(error, CollectionError) for error in self.collection_errors)
- assert self.collected_count > 0
- assert self.deselected_count > 0

COVERAGE

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

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the requirement marker is executed without causing any errors.

Why Needed: This test prevents a potential bug where the requirement marker causes an error in the plugin's execution.

Key Assertions:

- The `requirement_marker` function should not throw any exceptions or raise an error.
- The `requirement_marker` function should not modify the plugin's state or behavior in a way that would cause errors.
- The `requirement_marker` function should only be executed when necessary, without causing unnecessary execution of the plugin.

COVERAGE

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

PASSED `tests/test_plugin_integration.py::TestReportGeneration::test_report_writer_integration` 33ms 6

AI ASSESSMENT

Scenario: Verify that the report writer integrates with pytest_llm_report models correctly and generates a full report.

Why Needed: This test prevents regression or bugs in the integration of the report writer with pytest_llm_report models, ensuring consistent report generation.

Key Assertions:

- The report writer should create a JSON file named 'report.json' containing the total number of tests and passed tests.
- The report writer should include all test nodes in the HTML output.
- The report writer should verify that the correct files are generated for each test node (e.g., 'test_a.py' for test_a.py::test_pass).
- The report writer should correctly format the JSON data with the total number of tests and passed tests.
- The report writer should correctly format the HTML output with all test nodes included.
- The report writer should not fail to generate a full report even if there are no failed tests.
- The report writer should handle errors properly, including passing an error message for failed 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	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 skip tests when the plugin is disabled.

Key Assertions:

- mock_report.session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_collectreport(mock_report).session.config.stash.get.assert_calledWith(_enabled_key, False)
- mock_report.session.config.stash.get.return_value == False
- pytest_collectreport(mock_report) is not called with _enabled_key=False
- mock_report.session.config.stash.get.return_value is not False
- _enabled_key is correctly set to 'collectreport' in the mock report's session config
- pytest_collectreport does not raise an exception when disabled

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	10 lines (ranges: 387-388, 391, 395-397, 408-409, 415-416)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_enabled` 2ms 2

AI ASSESSMENT

Scenario: Test that collectreport calls collector when enabled.

Why Needed: This test prevents a potential bug where the plugin does not call the collector even if collectreport is enabled.

Key Assertions:

- The `pytest_collectreport` function should be able to find and stash the `collector` mock object.
- The `handle_collection_report` method of the `mock_collector` object should be called with the `mock_report` argument.
- The `stash_get` function should return `True` when it encounters `_enabled_key` or `_collector_key` keys in the session configuration.
- The `handle_collection_report` method should call `handle_collection_report` on the `mock_collector` object with the `mock_report` argument.
- The `stash_get` function should not be called for other keys in the session configuration.
- The `handle_collection_report` method of the `mock_collector` object should be able to handle a valid collection report.
- The `stash_get` function should return `None` when it encounters `_enabled_key` or `_collector_key` keys that are not present in the session 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/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: Verify that the `pytest_collectreport` function behaves correctly when no session is available.

Why Needed: Prevents a potential bug where the collect report skips due to an unavailable session.

Key Assertions:

- The `session` attribute of the mock report object is not set.
- The `pytest_collectreport` function does not raise an exception when called with a mock report that lacks a session attribute.
- The `pytest_collectreport` function correctly skips the collect report without raising an error when no session is available.
- The `session` attribute of the mock report object is not set after calling `pytest_collectreport`.
- The `pytest_collectreport` function does not raise an exception when called with a mock report that lacks a session attribute.
- The `pytest_collectreport` function correctly skips the collect report without raising an error when no session is available.
- The `session` attribute of the mock report object is set to None after calling `pytest_collectreport`.
- The `pytest_collectreport` function does not raise an exception when called with a mock report that lacks a session 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/plugin.py`	8 lines (ranges: 387-388, 391, 395-397, 408, 412)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_session_none` 1ms 2

AI ASSESSMENT

Scenario: Verify that `pytest_collectreport` does not raise an error when the session is set to `None`.

Why Needed: Prevent a potential bug where `pytest_collectreport` raises an exception when the session is None, causing test failures.

Key Assertions:

- The function `pytest_collectreport` should not be called with a `session` argument that is `None`.
- A `pytest_collectreport` instance without a `session` attribute should not raise an error.
- The `session` attribute of the `pytest_collectreport` instance should remain unchanged when set to `None`.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`8 lines (ranges: 387-388, 391, 395-397, 408, 412)`

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_llm_enabled_warning` 3ms 3

AI ASSESSMENT

Scenario: Test that LLM enabled warning is raised when using the Ollama LLM report provider.

Why Needed: This test prevents a potential bug where the LLM provider 'ollama' is enabled and raises an error in Pytest.

Key Assertions:

- The `pytest_llm_report_provider` option should be set to `'ollama'`.
- The `llm_report_provider` option should not be set to a value that is not supported (e.g. `'other'`).
- The `llm_report_html`, `llm_report_json`, and `llm_report_pdf` options should be set to `None`.
- The `llm_evidence_bundle`, `llm_dependency_snapshot`, `llm_requests_per_minute`, `llm_aggregate_dir`, `llm_aggregate_policy`, `llm_aggregate_run_id`, and `llm_aggregate_group_id` options should not be set. However, the option `llm_max_retries` is still set to `None`.
- The `llm_max_retries` option should be set to a valid value (e.g. `5`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	44 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	29 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_validation_errors` 3ms 3

AI ASSESSMENT

Scenario: Test that validation errors raise UsageError when invalid configuration is provided.

Why Needed: This test prevents a potential bug where the plugin does not handle invalid configuration properly and raises a UsageError instead of providing informative error messages.

Key Assertions:

- mock_config.getini.side_effect should return `None` for all keys in `ini_values` when calling `getini` on mock_config.
- mock_config.option.llm_report_html should be `None` when called `None` on mock_config.option_llm_report_html.
- mock_config.option.llm_report_json should be `None` when called `None` on mock_config.option_llm_report_json.
- mock_config.option.llm_report_pdf should be `None` when called `None` on mock_config.option_llm_report_pdf.
- mock_config.option.llm_evidence_bundle should be `None` when called `None` on mock_config.option_llm_evidence_bundle.
- mock_config.option.llm_dependency_snapshot should be `None` when called `None` on mock_config.option_llm_dependency_snapshot.
- mock_config.option.llm_requests_per_minute should be `None` when called `None` on mock_config.option_llm_requests_per_minute.
- mock_config.option.llm_aggregate_dir should be an empty string when called `None` on mock_config.option_llm_aggregate_dir.
- mock_config.option.llm_aggregate_policy should be an empty string when called `None` on mock_config.option_llm_aggregate_policy.
- mock_config.option.llm_aggregate_run_id should be an empty string when called `None` on mock_config.option_llm_aggregate_run_id.
- mock_config.option.llm_aggregate_group_id should be an empty string when called `None` on mock_config.option_llm_aggregate_group_id.
- mock_config.option.llm_max_retries should be an integer value greater than 0 when called `None` on mock_config.option_llm_max_retries.
- mock_config.rootpath should be a valid Path object when calling `getini` on mock_config.rootpath.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	43 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 248, 251-253,

	255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	25 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-199, 387-388, 391, 395-397)

PASSED tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_worker_skip 1ms 2

AI ASSESSMENT

Scenario: Test that configure skips on xdist workers and ensures addinivalue_line is not called before worker check.

Why Needed: This test prevents a regression where configure might skip the worker input due to an unhandled marker.

Key Assertions:

- mock_config.addinivalue_line was not called before worker check.
- mock_config.workerinput was set correctly.
- pytest_configure() was not called with mock_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/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 is triggered when Config.load is missing and no other fallbacks are available.

Why Needed: Prevents a potential bug where the plugin fails to configure due to an unhandled Config.load error, potentially leading to unexpected behavior or errors in downstream tests.

Key Assertions:

- Mocking `Config.load` with `None` returns `None`
- The `validate()` method of `mock_cfg` returns an empty list
- The `load_config()` function from `pytest_llm_report` is called with `mock_cfg` as argument
- The `load_config()` function does not raise any exceptions when called with `mock_cfg` as argument
- The `option.llm_max_retries` and `option.llm_report_html` are set to `None` in the mock configuration
- No other fallbacks (e.g. `pytest_llm_report.options.load_config`) are used, preventing the plugin from configuring

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	29 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_all_ini_options` 2ms 3

AI ASSESSMENT

Scenario: Test loading all INI options from the plugin configuration.

Why Needed: Prevents regression where CLI options are not set for 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.

Key Assertions:

- The correct provider is set to 'ollama'.
- The correct model is set to 'llama3.2'.
- The correct context mode is set to 'complete'.
- The correct number of requests per minute is set to 10.
- The report HTML and JSON files are set correctly.
- LLM PDF, EVIDENCE BUNDLE, DEPENDENCY SNAPSHOT, AGGREGATE_DIR, AGGREGATE_POLICY, AGGREGATE_RUN_ID, AND AGGREGATE_GROUP_ID are all 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 when using pytest_llm_report.

Why Needed: This test prevents a regression where the CLI options override INI options, causing unexpected behavior in the plugin.

Key Assertions:

- The `llm_report_html` option is set to 'cli.html' instead of 'ini.html'.
- The `llm_report_json` option is set to 'cli.json' instead of 'ini.json'.
- The `llm_report_pdf` option is set to 'cli.pdf' instead of 'ini.pdf'.
- The `llm_evidence_bundle` option is set to 'bundle.zip' instead of 'ini.evidence_bundle'.
- The `llm_dependency_snapshot` option is set to 'deps.json' instead of 'ini.dependency_snapshot'.
- The `llm_requests_per_minute` option is set to 20 instead of the expected value.
- The `aggregate_dir` option is set to '/agg' instead of the expected value.
- The `aggregate_policy` option is set to 'merge' instead of the expected value.
- The `aggregate_run_id` option is set to 'run-123' instead of the expected value.
- The `aggregate_group_id` option is set to 'group-abc' instead of 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	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 key.

Why Needed: This test prevents a regression where the plugin's terminal summary does not skip when it is disabled.

Key Assertions:

- mock_config.stash.get.assert_called_once_with(_enabled_key, True)
- mock_config.stash.get.return_value == False
- result is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	9 lines (ranges: 238, 242-243, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::test_terminal_summary_worker_skip` 1ms 2

AI ASSESSMENT

Scenario: Test that terminal summary skips on xdist worker when the workerid matches the config.

Why Needed: To prevent a regression where the plugin does not skip terminal summaries when an xdist worker is present.

Key Assertions:

- The function pytest_terminal_summary() should return None if the workerid in the mock_config matches the workerid of the xdist worker.
- The function pytest_terminal_summary() should not perform any action (i.e., return a value other than None) when the workerid in the mock_config matches the workerid of the xdist worker.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 238-239, 387-388, 391, 395-397)

PASSED tests/test_plugin_maximal.py::TestPluginMaximal::testload_config 3ms 3

AI ASSESSMENT

Scenario: Test config loading from pytest objects (CLI + INI) to ensure correct HTML output.

Why Needed: This test prevents a bug where the plugin does not display the correct HTML report even if the option is set correctly in the configuration.

Key Assertions:

- The `report_html` attribute of the loaded config object should be set to 'out.html'.
- The `llm_report_html` option should have an effect on the reported HTML output.
- The `rootpath` option should not affect the reported HTML output.
- The `getini` method of the mock_config object should return None for all keys when called.
- The `load_config` function should correctly load the config from the mock_config object and set the expected value for `report_html`.
- The `load_config` function should not raise an exception if the option is not present in the configuration.
- The `load_config` function should return a valid config object with the correct 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/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: Prevents a potential regression where the plugin does not report test results correctly when makereport is disabled.

Key Assertions:

- mock_item.config.stash.get() returns False
- mock_call() should be called with mock_outcome
- mock_outcome.get_result() should return a MagicMock object
- gen.send(mock_outcome) should not raise an exception when StopIteration is reached

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	7 lines (ranges: 387-388, 391-392, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_enabled` 2ms 2

AI ASSESSMENT

Scenario: Test makereport calls collector when enabled.

Why Needed: Prevents a potential bug where the plugin does not call the collector even though it is enabled.

Key Assertions:

- The `pytest_runtest_makereport` function should be able to find and stash the `_collector_key` key in the mock item's config.
- The `pytest_runtest_makereport` function should be able to find and return the `mock_collector` object from the stash.
- The `handle_runtest_logreport` method of the `mock_collector` object should be called with the `mock_report` object as an argument.
- The `get_result` method of the `mock_outcome` object should return a successful result (i.e., `None`) when the plugin is enabled.
- The `stash_get` function should return `True` for the `_enabled_key` key in the mock item's config.
- The `stash_get` function should not return any value for other keys.
- The `send` method of the `mock_outcome` object should be called with a successful result (i.e., `None`) when the plugin is enabled.
- The `handle_runtest_logreport` method of the `mock_collector` object should be called with the `mock_report` object as an argument.

COVERAGE

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

Why Needed: This test prevents a regression where the plugin's collection_finish behavior changes unexpectedly when collection_finish is disabled.

Key Assertions:

- The pytest_collection_finish function should not be called with _enabled_key as False.
- The pytest_collection_finish function should not be called with _enabled_key as True.
- The pytest_collection_finish function should not have been called on the mock session.
- The pytest_collection_finish function should not have had a return value of False.
- The pytest_collection_finish function's config stash.get method should have returned None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 387-388, 391, 395-397, 431-432)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_enabled` 2ms 2

AI ASSESSMENT

Scenario: Test that `pytest_collection_finish` calls `collector_key` when collection_finish is enabled.

Why Needed: Prevents a potential bug where the collector is not called when collection finish is enabled, potentially leading to incorrect data being collected.

Key Assertions:

- The `collector_key` of the session items should be called with the correct arguments.
- The `collector_key` should be called with the `mock_collector` instance as an argument.
- The `collector_key` should not be called with a default value.
- The `handle_collection_finish` method of the `collector_key` should be called once with the `mock_session.items` list.
- The `collector_key` should not call its own `handle_collection_finish` method without an argument.
- The `collector_key` should raise an exception if it is not properly configured.
- The `collector_key` should not call any other methods or attributes on the session items when called with a 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/plugin.py	10 lines (ranges: 387-388, 391, 395-397, 431, 435-437)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_disabled` 1ms 2

AI ASSESSMENT

Scenario: Test that sessionstart skips when disabled and checks enabled status.

Why Needed: Prevents a potential issue where the plugin is not properly configured or is disabled by mistake.

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 be 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 initialized due to an unenabled sessionstart.

Key Assertions:

- The `_collector_key` should be present in `mock_stash`.
- The `_start_time_key` should be present in `mock_stash`.
- A mock stash that supports both get() and [] operations should be created.
- The collector should be created after the sessionstart is initialized.
- The collector's key should match `_collector_key` from `mock_stash`.
- The start time of the collection should be recorded 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 prevents a potential bug where it does not add all required arguments to the parser.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	99 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption_ini` 2ms 2

AI ASSESSMENT

Scenario: Test pytest_addoption adds INI options (lines 13-34) to verify that the plugin correctly adds necessary configuration files.

Why Needed: This test prevents a regression where the plugin does not add necessary INI options for pytest, potentially causing issues with report generation.

Key Assertions:

- llm_report_html is added as an option
- llm_report_json is added as an option
- llm_report_max_retries is added as an option

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	99 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_coverage_calculation` 4ms 3

AI ASSESSMENT

Scenario: Test coverage percentage calculation logic for terminal summary.

Why Needed: Prevents regression in coverage reporting when using the `terminal_summary` plugin.

Key Assertions:

- The `report_html` option is set to 'out.html'.
- The `stash` dictionary contains both `_enabled_key` and `_config_key`.
- The `MockStash` class is used to mock the stash data. The `mock_config.stash` attribute is assigned this instance.
- The `CoverageMapper` class is mocked with a return value of 85.5 for the `report` method.
- The `pytest_terminal_summary` function is called with a mock `MockStash` object and an integer argument (0).
- The `mock_cov_cls.return_value` attribute is set to the `mock_cov` instance, which has been mocked to return 85.5 for the `report` method.
- The `mock_cov.load.assert_called_once()` assertion checks that the `load` method was called once with no arguments.
- The `mock_cov.report.assert_called_once()` assertion checks that the `report` method was called once with no arguments.
- The `pytest_terminal_summary` function is called again, which should not have any additional arguments (1).

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	58 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-305, 307, 309-315, 317-318, 331-332, 337-338, 365-375, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_llm_enabled` 3ms 3

AI ASSESSMENT

Scenario: Test that terminal summary with LLM enabled runs correctly and reports the correct provider.

Why Needed: This test prevents regression in the plugin's behavior when LLM is enabled.

Key Assertions:

- The `pytest_terminal_summary_llm_enabled` function should be called with a valid configuration.
- The provided configuration should match the expected one.
- The correct provider should be reported in the terminal summary.
- The report HTML file should contain the correct information.
- The coverage map should not be affected by LLM enabled.
- The annotate tests should run successfully without any errors.
- The `pytest_terminal_summary_llm_enabled` function should return a mock object with the expected arguments.
- The `mock_annotate.call_args[0][1] == cfg` assertion should pass when the provided configuration is correct.

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 that the terminal summary creates a collector if it is missing.

Why Needed: Prevents regression and ensures proper behavior when no collector is specified.

Key Assertions:

- The `pytest_terminal_summary` function should create a `MockStash` instance with `_enabled_key` set to `True` and `_config_key` set to the provided configuration object.
- The `stash` attribute of the mock stash instance should be set to the provided stash instance.
- The `mock_config.stash` assignment should update the stash instance with the provided stash instance.
- The `pytest_terminal_summary` function should call the `ReportWriter` patcher with a valid `MockTerminalReporter` instance and return value.
- The `coverage_map` mapper should not be called with any coverage data when creating a mock stash instance.
- The `_enabled_key` key in the mock stash instance should be set to `True` after creation.
- The `_config_key` key in the mock stash instance should be set to the provided configuration object after creation.
- A new mock terminal reporter instance should be created with the provided arguments (0, mock_config).

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	45 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_with_aggregation` 2ms 3

AI ASSESSMENT

Scenario: Test terminal summary with aggregation enabled.

Why Needed: This test prevents regression in the case where aggregation is enabled and a stash that supports both get() and [] is created.

Key Assertions:

- The `aggregate_dir` attribute of the configuration object should be set to `/agg` when aggregation is enabled.
- The `stash` attribute of the configuration object should contain an `_enabled_key` with value `True` when aggregation is enabled.
- The `aggregate` method of the Aggregator instance should return a report when aggregation is enabled.
- The `write_json` and `write_html` methods of the ReportWriter instance should be called once when aggregation is enabled.
- The stash created by creating a proper stash that supports both get() and [] should contain an `_enabled_key` with value `True` when aggregation is enabled.
- The Aggregator instance should have an aggregate method that returns a report when aggregation is enabled.
- The ReportWriter instance should write JSON and HTML files when aggregation is enabled.
- The stash created by creating a proper stash that supports both get() and [] should not contain any other attributes or methods when aggregation is enabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 that a coverage calculation error prevents the test from running.

Why Needed: This test verifies that the `pytest_terminal_summary` function raises a warning when it encounters an error during coverage calculation.

Key Assertions:

- The `pytest_terminal_summary` function should raise a `UserWarning` with message 'Failed to compute coverage percentage'.
- The `pytest_terminal_summary` function should not be able to calculate the coverage percentage if the disk is full.
- The `pytest_terminal_summary` function should log an error message when it encounters an error during coverage calculation.
- The `pytest_terminal_summary` function should raise a warning with the correct message when it encounters an error during coverage calculation.
- The `pytest_terminal_summary` function should not be able to calculate the coverage percentage if the disk is full or an error occurs during coverage calculation.
- The `pytest_terminal_summary` function should log an error message when it encounters an error during coverage calculation and raise a warning with the correct message.

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.

Why Needed: This test prevents regression in case of unbalanced contexts, where only one module's dependencies are included.

Key Assertions:

- The 'utils.py' file is present in the assembled context.
- The 'def util()' function is found in the 'utils.py' file within the assembled context.
- Only 'test_a.py' and its contents are included in the assembled context.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	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_1` function in `test_a.py`

Why Needed: This test prevents regression when the `complete` mode is used with a non-existent file.

Key Assertions:

- The `source` variable should contain the contents of `test_a.py::test_1`.
- The `context` variable should be an empty string if `test_a.py::test_1` does not exist.
- If `test_a.py::test_1` exists, the `source` and `context` variables should both contain the contents of that function.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 the ContextAssembler to assemble a minimal context for a test file.

Why Needed: This test prevents regression when assembling contexts with minimal llm_context_mode.

Key Assertions:

- The 'test_1' function is present in the source code of the test file.
- The context object returned by the ContextAssembler has an empty dictionary.
- The 'test_1' function is executed successfully without raising any exceptions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	30 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 116)

PASSED `tests/test_prompts.py::TestContextAssembler::test_balanced_context_limits` 1ms 4

AI ASSESSMENT

Scenario: Test the ContextAssembler to ensure it can assemble a file with a balanced context limit and handle long content without truncation.

Why Needed: This test prevents a potential bug where the ContextAssembler fails to assemble files that exceed the specified context limit, potentially leading to incorrect results or errors.

Key Assertions:

- The assembler should be able to assemble the file `f1.py` with a 'balanced' context limit.
- The assembler should not truncate the content of `f1.py` when its length exceeds 40 bytes.
- The assembler should report an error message indicating that the file has been truncated when its length exceeds 40 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/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 correct behavior when a non-existent file is provided to `_get_test_source`.

Why Needed: This test prevents a potential bug where an empty string is returned for files that do not exist.

Key Assertions:

- The function should return an empty string for a non-existent file.
- The function should raise a `FileNotFoundError` exception when trying to access the file.
- The function should report an error message indicating that the 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/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 LLM context.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly includes certain files in the LLM context.

Key Assertions:

- asserts that 'test.pyc' is excluded because it's a Python bytecode file and the config includes glob patterns for .pyc files
- asserts that 'secret/key.txt' is excluded because it contains sensitive information and the config excludes secret directories
- asserts that 'public/readme.md' is not excluded because it does not contain any sensitive information

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Testing consecutive lines in the `compress_ranges` function.

Why Needed: This test prevents regression where consecutive lines are not compressed correctly.

Key Assertions:

- The function should return a string with range notation for consecutive lines (e.g., '1-3').
- The function should handle cases where there is only one line (e.g., [1]).
- The function should ignore non-consecutive lines when compressed.
- The function should not compress empty lists or lists containing only zeros.
- The function should handle lists with negative numbers correctly (e.g., [-3, 1]).

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/ranges.py`	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

PASSED `tests/test_ranges.py::TestCompressRanges::test_duplicates` 1ms 3

AI ASSESSMENT

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

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/ranges.py`	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

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

AI ASSESSMENT

Scenario: Testing the `compress_ranges` function with an empty input list

Why Needed: This test prevents a potential bug where an empty list is not correctly compressed.

Key Assertions:

- The function should return an empty string for an empty input list.
- The function should handle empty lists without raising any exceptions or errors.
- The function should preserve the original order of elements in the input list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	2 lines (ranges: 29-30)

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

AI ASSESSMENT

Scenario: Test the function when given a mixed range of numbers.

Why Needed: This test prevents regression in cases where ranges are mixed with single values.

Key Assertions:

- The output should be '1-3, 5, 10-12, 15'.
- The output should contain all the given ranges.
- Each range should have a start value that is less than or equal to its end value.
- The function should handle single values correctly as well.
- No duplicate ranges are allowed in the output.
- The order of ranges matters (e.g., '1-3, 5' comes before '10-12').
- The function should raise an error if a range is not valid (e.g., start value greater than end value).

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)`

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

AI ASSESSMENT

Scenario: Test 'non_consecutive_lines' verifies that non-consecutive lines are correctly compressed into a single range.

Why Needed: This test prevents regression in the compress_ranges function where non-consecutive lines are not being properly handled.

Key Assertions:

- The input list should be sorted and then comma-separated.
- The resulting string should contain only one comma.
- All numbers in the range should be consecutive.
- No leading or trailing commas should be present.
- Compressed ranges with non-consecutive lines should not exceed 2 characters.
- Compressed ranges with more than 3 numbers should not be split into multiple parts.
- 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' scenario verifies that a single line of code does not utilize the range notation.

Why Needed: This test prevents regression where a single-line function is incorrectly compressed using range notation.

Key Assertions:

- assert compress_ranges([5]) == '5'
- assert isinstance(compress_ranges([5]), str)
- assert len(compress_ranges([5])) == 1
- assert compress_ranges([]) is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	10 lines (ranges: 29, 33, 35-37, 39, 50, 52, 65-66)

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

AI ASSESSMENT

Scenario: tests/test_ranges.py::TestCompressRanges::test_two_consecutive

Why Needed: This test prevents a regression where two consecutive numbers are not compressed to range notation.

Key Assertions:

- The function should return the correct string representation of the compressed range (e.g. '1-2') for two consecutive numbers.
- The function should handle cases where the input list contains only one element correctly.
- The function should raise an error when given a non-list input.
- The function should not raise an error when given a list with less than two elements.
- The function should return the correct string representation of the compressed range for negative numbers.
- The function should handle cases where the input list contains duplicate values correctly.
- The function should raise an error when given a list that is empty or contains only one element.
- The function should not raise an error when given a list with only two elements.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 potential bugs where the function does not handle unsorted ranges properly.

Key Assertions:

- The function should return a string with the correct range notation for unsorted input.
- The function should sort the input ranges before returning them.
- The function should raise an error if the input is not sorted.
- The function should preserve the original order of equal elements in the input.
- The function should handle empty input correctly.
- The function should return a string with the correct range notation for unsorted input even when there are multiple ranges.
- The function should raise an error if the input is not a list or other iterable.
- The function should preserve the original order of equal elements in the input.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)

PASSED `tests/test_ranges.py::TestExpandRanges::test_empty_string` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that an empty string expands to an empty list.

Why Needed: This test prevents a potential bug where the function does not handle empty input strings correctly.

Key Assertions:

- Input: An empty string
- Expected Output: An empty list
- Expand ranges on empty string: True
- No expansion on empty string: False
- Empty string should produce an empty list of tuples
- Empty string should produce a list with one element (an empty tuple)
- Empty string should produce a list with two elements (the original input and an empty tuple)

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`2 lines (ranges: 81-82)`

PASSED tests/test_ranges.py::TestExpandRanges::test_mixed 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `expand_ranges` function correctly handles mixed ranges and singles.

Why Needed: This test prevents regression when the input contains a mix of single values and ranges.

Key Assertions:

- The function should return a list containing all specified numbers.
- The function should handle ranges by extracting their start and end values.
- The function should correctly handle overlapping ranges.
- The function should ignore empty strings in the input.
- The function should preserve the order of single values.
- The function should handle invalid or malformed input (e.g., non-string, non-integer).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	11 lines (ranges: 81, 84-91, 93, 95)

PASSED tests/test_ranges.py::TestExpandRanges::test_range 1ms 3

AI ASSESSMENT

Scenario: The 'expand_ranges' function is expected to expand a range of numbers into a list.

Why Needed: This test prevents a potential bug where the function does not correctly handle ranges with negative numbers or zero.

Key Assertions:

- The function should return a list containing all numbers in the range (1, 3).
- The function should include both 1 and 2 in the returned list.
- The function should exclude 0 from the returned list if it is present in the input range.
- The function should handle ranges with negative numbers correctly (e.g., '-1-3').
- The function should not return an empty list for invalid input (e.g., 'abc-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	10 lines (ranges: 81, 84-91, 95)

PASSED tests/test_ranges.py::TestExpandRanges::test_roundtrip 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `expand_ranges` function correctly expands a range of numbers back to its original form after being compressed.

Why Needed: This test prevents a potential bug where the `expand_ranges` function does not correctly expand ranges when they are compressed.

Key Assertions:

- The expanded list should be equal to the original list.
- The first element of the expanded list should still be 1.
- The second element of the expanded list should still be 2.
- The third element of the expanded list should still be 3.
- The fourth element of the expanded list should still be 5.
- The fifth element of the expanded list should still be 10.
- The sixth element of the expanded list should still be 11.
- The seventh element of the expanded list should still be 12.
- The eighth element of the expanded list should still be 15.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	27 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67, 81, 84-91, 93, 95)

PASSED `tests/test_ranges.py::TestExpandRanges::test_single_number` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `expand_ranges` function returns a list containing only one element when given a single number.

Why Needed: This test prevents a potential bug where the function incorrectly expands ranges for single numbers, potentially leading to incorrect results or unexpected behavior.

Key Assertions:

- The input string should be '5' (a single digit).
- The output list should contain only one element: `[5]`.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/ranges.py`	7 lines (ranges: 81, 84-87, 93, 95)

PASSED `tests/test_render.py::TestFormatDuration::test_milliseconds` 1ms 3

AI ASSESSMENT

Scenario: Test that the `format_duration` function correctly formats durations for milliseconds less than 1 second.

Why Needed: This test prevents a regression where the function does not correctly handle durations in milliseconds less than 1 second.

Key Assertions:

- The function should return '500ms' when given an input of 0.5 seconds.
- The function should return '1ms' when given an input of 0.001 seconds.
- The function should return '0ms' when given an input of 0.0 seconds.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/render.py`	2 lines (ranges: 65, 67)

PASSED tests/test_render.py::TestFormatDuration::test_seconds 1ms 3

AI ASSESSMENT

Scenario: Test the `format_duration` function with a duration of exactly 1 second.

Why Needed: This test prevents a regression where the function returns incorrect results for durations less than or equal to 1 second.

Key Assertions:

- The output should be '1.23s' when the input is 1.23 seconds.
- The output should be '60.00s' when the input is 60 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	2 lines (ranges: 65-66)

PASSED tests/test_render.py::TestOutcomeToCssClass::test_all_outcomes 1ms 3

AI ASSESSMENT

Scenario: Test that all outcomes map to CSS classes.

Why Needed: Prevents regression in rendering of test results.

Key Assertions:

- The `outcome_to_css_class` function should correctly map each outcome to a CSS class.
- The function should handle special cases like 'xfailed' and 'error' outcomes.
- The function should return the correct CSS class for each outcome.
- The function should not throw any exceptions when handling invalid inputs.
- The function should preserve the original value of the input string.
- The function should use a consistent naming convention for the CSS classes.
- The function should be able to handle different types of outcomes (e.g. strings, integers).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	8 lines (ranges: 79-85, 87)

PASSED tests/test_render.py::TestOutcomeToCssClass::test_unknown_outcome 1ms 3

AI ASSESSMENT

Scenario: tests/test_render.py::TestOutcomeToCssClass::test_unknown_outcome

Why Needed: This test prevents a regression where the `outcome_to_css_class` function returns an unknown outcome and does not set a default CSS class.

Key Assertions:

- The function `outcome_to_css_class('unknown')` should return 'outcome-unknown'.
- The function `outcome_to_css_class('unknown')` should raise an AssertionError with message 'Unknown 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: Tests the rendering of a basic report with fallback HTML.

Why Needed: This test prevents regression that may occur when the rendered report contains missing or incomplete information.

Key Assertions:

- The document type declaration should be present in the HTML.
- The title of the report should be 'Test Report'.
- The passed node should have a text content equal to 'PASSED'.
- The failed node should have a text content equal to 'FAILED'.
- The plugin version and repository version should be displayed correctly.
- The duration of each test result should be reported 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/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 to ensure that the rendered HTML includes the specified file and line count.

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

Key Assertions:

- The 'src/foo.py' file should be present in the rendered HTML.
- The rendered HTML should include at least 5 lines (1-5).
- The number of lines in the rendered HTML should match the total line count reported by ReportRoot.
- The coverage report for the 'test::foo' test should include the specified file and line count.
- The rendered HTML should contain the expected number of lines based on the coverage report.
- The coverage report should not be empty or null.
- The file path in the rendered HTML should match the specified file path.
- The line ranges in the coverage report should match the specified line range (1-5).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	52 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-129, 131-132, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_llm_annotation` 1ms 3

AI ASSESSMENT

Scenario: Test renders LLM annotations for fallback HTML.

Why Needed: This test prevents the rendering of LLM annotations that could be used to bypass authentication.

Key Assertions:

- The 'Tests login flow' scenario should be present in the rendered HTML.
- The 'Prevents auth bypass' reason should be present in the rendered HTML.
- The LlmAnnotation nodeid should be included in the rendered HTML.
- The LLmAnnotation scenario should be included in the rendered HTML.
- The LLmAnnotation why_needed message should be included in the rendered HTML.
- The ReportRoot summary should include a total of 1 test and a passed of 1 test.
- The TestCaseResult nodeid should match 'test::foo'.
- The TestCaseResult outcome should be '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)
src/pytest_llm_report/render.py	54 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-127, 131-134, 136-137, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_source_coverage` 1ms 3

AI ASSESSMENT

Scenario: Test verifies that the source coverage summary is included in the rendered HTML.

Why Needed: This test prevents regression where the source coverage summary is not displayed correctly due to missing or incorrect data.

Key Assertions:

- The 'Source Coverage' section should be present in the rendered HTML.
- The 'src/foo.py' file path should be included in the 'Source Coverage' section.
- The percentage of covered code (80.0%) should be displayed correctly in the 'Source Coverage' section.
- The ranges where missed code is reported ('1-4, 6-8') should be present in the 'Source Coverage' section.
- The ranges where missed code is not reported ('5, 9-10') should not be present in the 'Source Coverage' section.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/render.py`	63 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-127, 131-132, 141-142, 145-153, 158-164, 166-172, 177, 192, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_xpass_summary` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the 'XFailed' and 'XPassed' summary entries are included in the rendered HTML.

Why Needed: This test prevents a regression where the xfailed/xpassed summary is not displayed correctly.

Key Assertions:

- The string 'XFailed' should be present in the rendered HTML.
- The string 'XPassed' should be present in the rendered HTML.
- Both 'XFailed' and 'XPassed' should be 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 'different_content' verifies that computing the SHA-256 of different strings produces different hashes.

Why Needed: This test prevents a potential bug where two different strings could produce the same hash, potentially leading to incorrect reporting or analysis.

Key Assertions:

- The function `compute_sha256()` should return different values for different input bytes.
- The function `compute_sha256()` should raise an error if the input is not a bytes object.
- The function `compute_sha256()` should handle non-string inputs correctly and produce a hash.
- The function `compute_sha256()` should be able to handle large strings without running out of memory.
- The function `compute_sha256()` should support different character encodings (e.g., UTF-8, ASCII).
- The function `compute_sha256()` should raise an error if the input is not a valid bytes object.
- The function `compute_sha256()` should be able to handle binary data correctly and produce a hash.
- The function `compute_sha256()` should support different encoding schemes (e.g., UTF-8, ASCII).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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 'Empty bytes should produce consistent hash'.

Why Needed: This test prevents a potential bug where different inputs to `compute_sha256` result in inconsistent hashes.

Key Assertions:

- The output of `compute_sha256(b'')` and `compute_sha256(b'')` should be the same.
- The length of the output of `compute_sha256(b'')` should be 64 characters (the 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 information for the test case.

Why Needed: This test prevents a regression where the test case's metadata is missing or incorrect.

Key Assertions:

- The duration of the test should be correctly calculated.
- The pytest version should be present in the metadata.
- The plugin version and Python version should also be included in the metadata.
- The start time and end time should be correctly formatted as datetime objects.
- The exit code should be 0, indicating successful execution of the test case.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 does not accurately count all outcome types, leading to incorrect reporting of test results.

Key Assertions:

- The total number of outcomes should be equal to the sum of passed, failed, skipped, xfailed, and xpassed outcomes.
- The number of passed outcomes should match the expected value (1 in this case).
- The number of failed outcomes should also match the expected value (1 in this case).
- The number of skipped outcomes should be 0 since there are no skipped tests in this test.
- The number of xfailed and xpassed outcomes should also match the expected values (1 each in this case).
- The number of error outcomes should be 1, as specified in the TestCaseResult nodeid for outcome 'error'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	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 `total` count of tests is correctly calculated by summing up all passed and failed outcomes.

Why Needed: This test prevents a regression where the total count does not match the expected value when running multiple tests with different outcomes.

Key Assertions:

- The total number of tests should be equal to the sum of passed and failed outcomes.
- The total number of tests should be equal to 4 (2 passed, 1 failed, and 1 skipped).
- All test nodes should have a valid `outcome` attribute.
- All test nodes should have a valid `nodeid` attribute.
- All test nodes should have a valid `passed` or `failed` attribute.
- The `skipped` outcome should be represented as `None` in the summary.
- The `total` count should match the expected value when running multiple tests with different outcomes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	13 lines (ranges: 156-158, 312, 314-315, 317-322, 330)

PASSED tests/test_report_writer.py::TestReportWriter::test_create_writer 1ms 4

AI ASSESSMENT

Scenario: Test that the `ReportWriter` class initializes correctly and returns a reference to its internal configuration.

Why Needed: This test prevents regression where the `ReportWriter` instance is not properly initialized with the expected configuration.

Key Assertions:

- The `config` attribute of the `writer` object should be set to an instance of `Config`.
- The `warnings` attribute of the `writer` object should be an empty list.
- The `artifacts` attribute of the `writer` object should be an empty list.
- The `writer` object should return a reference to its internal configuration, i.e., `config`, when created.
- The `writer` object's attributes (`warnings`, `artifacts`) should not have any values assigned to them initially.

COVERAGE

src/pytest_llm_report/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 'ReportWriter::test_write_report_assembles_tests' verifies that the ReportWriter class writes a report with all tests.

Why Needed: This test prevents regression where the ReportWriter class does not write reports for all tests, potentially leading to missing information in test summaries.

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.
- All tests should have been included in the report.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	93 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_includes_coverage_percent` 6ms 4

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class writes a report with a total coverage percentage.

Why Needed: This test prevents regression where the coverage percentage is not included in the report.

Key Assertions:

- The `report.summary.coverage_total_percent` property should be equal to the provided `coverage_percent` value.
- The `writer.write_report()` method should create a new report with the specified `coverage_percent` value.
- The `report.summary` object should have a `coverage_total_percent` attribute that matches the provided value.
- The coverage percentage is calculated correctly by summing up all individual coverage percentages.
- The test does not fail when the coverage percentage is less than 100% (e.g., 0%).
- The test does not fail when the coverage percentage is exactly 100% (e.g., 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/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 regression where the report does not include source coverage information, which is crucial for understanding code quality and identifying areas of improvement.

Key Assertions:

- The length of `report.source_coverage` should be 1.
- The file path of `report.source_coverage[0]` should match 'src/foo.py'.
- All statements in the source coverage entry should be covered by the report.
- At least one statement out of the five missed statements should be covered by the report.
- The percentage of covered statements should be 87.5% or higher.
- Covered ranges should match '1-4, 6-7'.
- Missed ranges should not exceed '5'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	92 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_merges_coverage` 5ms 4

AI ASSESSMENT

Scenario: Test Report Writer should merge coverage into tests.

Why Needed: This test prevents a bug where the coverage is not merged correctly, causing reports to have incorrect coverage information.

Key Assertions:

- The length of the report's coverage for 'test1' should be 1 (i.e., all lines are covered).
- The file path of the first line in the coverage for 'test1' 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	94 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186-189, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_atomic_write_fallback` 6ms 5

AI ASSESSMENT

Scenario: Test that the report writer falls back to direct write if atomic write fails and a file exists.

Why Needed: This test prevents a regression where an atomic write attempt fails, but the report still gets written to a temporary location.

Key Assertions:

- The file `report.json` should exist at the specified path.
- Any warnings with code 'W203' should be present in the `report.json` 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	67 lines (ranges: 229-231, 233, 235, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	125 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506-507, 509-512, 515-516)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_creates_directory_if_missing` 6ms 5

AI ASSESSMENT

Scenario: Test verifies that a directory is created if it does not exist.

Why Needed: Prevents regression where the test fails due to an existing output directory.

Key Assertions:

- The 'subdir' directory should be created in the specified path.
- The 'report.json' file should be written to this directory.
- The 'tmp_path / subdir / report.json' path should exist after calling write_report() 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/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: Verify that the test captures a warning when attempting to create a non-existent directory.

Why Needed: This test prevents a potential bug where the report writer fails to capture warnings for directories that cannot be created.

Key Assertions:

- The `writer.warnings` list contains at least one warning with code 'W201' (Permission denied).
- The `writer.warnings` list does not contain any warnings with code other than 'W201'.
- The `writer.warnings` list is empty.
- Any of the warnings in `writer.warnings` have a non-zero code value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	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 potential regression where the `get_git_info` function fails to return expected values when encountering a git command failure, potentially leading to incorrect report generation or downstream issues.

Key Assertions:

- The `get_git_info` function should return `None` for both SHA and dirty flags if the git command fails.
- The `get_git_info` function should not raise an exception when encountering a git command failure.
- The test should fail when the `get_git_info` function returns `None` for both SHA and dirty flags.
- The test should pass when the `subprocess.check_output` mock returns an exception with the correct error message.
- The test should verify that the returned values are consistent across different git command failures (e.g., 'git not found' vs. 'git: not found').
- The test should handle the case where the git command is not found but its output is available (e.g., when running `git --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/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 ReportWriter creates an HTML file with expected content.

Why Needed: This test prevents a regression where the report writer does not create an HTML file as expected, potentially leading to incorrect reporting or output.

Key Assertions:

- The report.html file should exist at the specified path.
- The report.html file should contain the expected content (tests 'test1', 'test2' and their respective outcomes).
- The report.html file should include the following text: 'PASSED', 'FAILED', 'Skipped', 'XFailed', 'XPassed', and 'Errors'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	115 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_html_includes_xfail_summary` 32ms 5

AI ASSESSMENT

Scenario: Test `test_write_html_includes_xfail_summary` verifies that the report includes xfail outcomes in the HTML summary.

Why Needed: This test prevents regression where the report does not include xfail outcomes, which can make it difficult to diagnose issues with failed tests.

Key Assertions:

- The 'XFAILED' and 'XPASSED' tags are present in the HTML summary.
- Both 'xfailed' and 'xpassed' tags are found in the HTML text.
- The 'XFailed' tag is also present in the HTML text.
- The 'XPASSED' tag is also present in the HTML text.
- All three 'XFAILED', 'XPASSED', and 'XFailed' tags are found 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 a JSON file is created with the report.

Why Needed: This test prevents a regression where the `ReportWriter` does not create a JSON file even when there are tests to write.

Key Assertions:

- The `report.json` file should be created in the specified path.
- At least one artifact should be tracked for the report.
- The number of artifacts tracked should be greater than zero.
- The `ReportWriter` should create a JSON file even when there are tests to write.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	78 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	117 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_pdf_creates_file` 34ms 5

AI ASSESSMENT

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	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 writing PDF reports with missing Playwright.

Why Needed: This test prevents the 'Warning: PDF output requires Playwright' message from appearing when using PDF output.

Key Assertions:

- The file 'report.pdf' should not exist.
- Any warnings related to 'PDF output requires Playwright' 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 of report writer output.

Why Needed: Prevents a potential issue where the report writer does not create the expected directory structure.

Key Assertions:

- The `tmp_dir / 'r.html'` path exists before attempting to write to it.
- Any warnings from the report writer are of type 'W202'.
- A directory named `r.html` is created in the temporary location.
- The `tmp_dir / 'r.html'` path does not exist after writing to 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	11 lines (ranges: 156-158, 470-477)

PASSED `tests/test_report_writer_coverage_v2.py::test_report_writer_metadata_skips` 9ms 5

AI ASSESSMENT

Scenario: Test verifies that report_writer_metadata_skips function prevents skipping of metadata when reports are disabled.

Why Needed: This test prevents a regression where the report writer skips metadata when reports are disabled, ensuring accurate reporting and compliance with requirements.

Key Assertions:

- The 'start_time' key should be present in the metadata dictionary.
- Metadata should contain an 'llm_model' key, which is expected to be None for this test case.
- The 'llm_model' value should not be provided in the metadata.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	36 lines (ranges: 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	67 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300)

PASSED tests/test_schemas.py::TestAnnotationSchema::test_from_dict_full 1ms 3

AI ASSESSMENT

Scenario: Test that `AnnotationSchema.from_dict` creates a valid annotation from a dictionary with all required fields.

Why Needed: Prevents regression in case of missing or malformed input data, ensuring the test suite remains stable and reliable.

Key Assertions:

- assert schema.scenario == 'Verify login'
- assert schema.why_needed == 'Catch auth bugs'
- assert schema.key_assertions == ['assert 200', 'assert token']
- assert schema.confidence == 0.95

COVERAGE

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

PASSED tests/test_schemas.py::TestAnnotationSchema::test_to_dict_full 1ms 3

AI ASSESSMENT

Scenario: Should convert to dictionary with all fields.

Why Needed: Prevent regression in authentication logic by ensuring correct conversion of schema to dictionary.

Key Assertions:

- assert data['scenario'] == 'Verify login'
- assert data['why_needed'] == 'Catch auth bugs'
- assert data['key_assertions'] == ['assert 200', 'assert token']
- assert data['confidence'] == 0.95

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	8 lines (ranges: 90-92, 94-98)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_report_created` 82ms 7

AI ASSESSMENT

Scenario: Verify that an HTML report is created and exists after running the test.

Why Needed: Prevents regression where the test fails to generate a report due to missing dependencies or configuration issues.

Key Assertions:

- The report file should be created at the specified path.
- The report file should contain the expected content.
- The report file should have an HTML structure.

COVERAGE

src/pytest_llm_report/collector.py	39 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)

src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	101 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_summary_counts_all_statuses` 118ms 7

AI ASSESSMENT

Scenario: Verify that HTML summary counts include all statuses.

Why Needed: This test prevents regression in case of unexpected pass or skip status.

Key Assertions:

- The 'XPassed' and 'XFailed' labels should be included in the summary.
- The 'Errors' label should also be included in the summary.
- All statuses (Total Tests, Passed, Failed, Skipped) should be counted correctly.

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` 71ms 7

AI ASSESSMENT

Scenario: The test verifies that a JSON report is created when the `--llm-report-json` option is used.

Why Needed: This test prevents regression where the report generation functionality fails to create a JSON file.

Key Assertions:

- The path to the generated report should exist.
- The report data should contain the expected schema version, summary statistics, and counts.
- The total count of passed tests should match the actual number of passed tests in the report.
- The total count of failed tests should match the actual number of failed tests in the report.
- All test names should be included in the report's 'summary' section.

COVERAGE

src/pytest_llm_report/collector.py	51 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-118, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 227-228, 230-236, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71,

	73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/report_writer.py	107 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_llm_annotations_in_report` 83ms 13

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report when a provider is enabled.

Why Needed: Prevent regressions by ensuring LLM annotations are present in reports.

Key Assertions:

- The 'scenario' key in the LLM annotation should match 'Checks the happy path'.
- The 'why_needed' key in the LLM annotation should contain a message that prevents regressions.
- The 'key_assertions' list within the LLM annotation should include 'asserts True'.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	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` 6.08s 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 'LLM error' should be present in the report HTML content.
- The 'boom' string should be present in the report HTML content.
- The presence of 'LLM error' and 'boom' strings should be verified by the test.

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 56ms 7

AI ASSESSMENT

Scenario: Test the LLM opt-out marker to ensure it correctly records LLM opt-out markers in the report.

Why Needed: This test prevents a regression where the LLM opt-out marker might not be recorded in the report, potentially leading to incorrect analysis results.

Key Assertions:

- The 'llm_opt_out' attribute of each test is set to True when running with LLM opt-out enabled.
- The number of tests in the report is exactly one.
- Each test has an 'llm_opt_out' attribute that is set to False (indicating LLM opt-out) if it was run with LLM opt-out enabled.
- The 'llm_opt_out' attribute is present in each test and its value is True.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181-182, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180-182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134,

	136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/report_writer.py	105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED tests/test_smoke_pytester.py::TestMarkers::test_requirement_marker 55ms 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, leading to incorrect reporting of tests.

Key Assertions:

- The `pytest.mark.requirement` decorator should be applied to the test function with the required requirements.
- The `requirement` parameter in the `@pytest.mark.requirement` decorator should match one of the provided requirement strings (REQ-001 or REQ-002).
- The `requirements` list in the test function's docstring should contain both 'REQ-001' and 'REQ-002'.

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 61ms 7

AI ASSESSMENT

Scenario: Verify that multiple xfailed tests are recorded in the report.

Why Needed: This test prevents regression by ensuring that all xfailed tests are properly reported and counted.

Key Assertions:

- The total number of xfailed tests is 2 (test_xfail_one) and test_xfail_two)
- All xfailed tests are correctly recorded in the report (outcome == 'xfailed')
- No other tests are incorrectly reported or missed (all outcomes match)

COVERAGE

src/pytest_llm_report/collector.py	47 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-116, 119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212, 216, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190,

	192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/report_writer.py	108 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321, 323-324, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED tests/test_smoke_pytester.py::TestOutcomes::test_skip_outcome 56ms 7

AI ASSESSMENT

Scenario: Test that skipped tests are recorded and reported correctly.

Why Needed: This test prevents a regression where the 'skip' marker is not properly recorded or reported in the report.

Key Assertions:

- The number of skipped tests should be 1 according to the report.
- The 'summary' key in the report data should contain the string 'skipped'.
- The value of the 'skipped' key in the report data should be 1.
- The test is skipped and recorded in the report.
- The test is not skipped and does not appear to be recorded in the report.
- The report path contains a file with the name 'report.json'.

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 57ms 7

AI ASSESSMENT

Scenario: Verifies that the test 'test_xfail' is marked as Xfailed and its outcome is recorded in the report.

Why Needed: This test prevents a regression where Xfailed tests are not correctly recorded in the report.

Key Assertions:

- The test 'test_xfail' should be marked as Xfailed by pytester.makepyfile.
- The assertion False in test_xfail should fail and record an outcome of 1 in the report.
- The value of data['summary']['xfailed'] in the report.json file should be 1.

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` 59ms 7

AI ASSESSMENT

Scenario: Test the functionality of parameterized tests to ensure they are recorded separately and their results are accurate.

Why Needed: This test prevents regression in the parametrization feature, ensuring that all test cases are correctly recorded and reported.

Key Assertions:

- The total number of successful tests should be equal to the expected number of tests (3).
- All tests passed successfully (x > 0).

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163-164, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167, 169-171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213,

	238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 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` 48ms 3

AI ASSESSMENT

Scenario: The CLI help text should include usage examples.

Why Needed: This test prevents a potential bug where the user is not informed about available usage examples in the CLI help text.

Key Assertions:

- The `--help` option is present in the CLI help text.
- The example usage is included at the end of the help message.
- The example usage is preceded by a comment indicating it's an example.
- The example usage is not truncated or removed from the output.
- The example usage is displayed correctly in the terminal.
- The example usage does not cause any issues with the test suite.
- The example usage is consistent across different platforms and environments.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 44ms 3

AI ASSESSMENT

Scenario: Test that LLM markers are registered and their presence is checked.

Why Needed: This test prevents a potential bug where the LLM markers are not properly registered or are not present in the output.

Key Assertions:

- The string 'llm_opt_out*' should be found in the stdout of the runpytest command.
- The string 'llm_context*' should be found in the stdout of the runpytest command.
- The string 'requirement*' should be found in the stdout of the runpytest command.

COVERAGE

src/pytest_llm_report/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` 51ms 3

AI ASSESSMENT

Scenario: A plugin is successfully registered using pytest11 and the LLM report is displayed.

Why Needed: This test prevents a potential issue where the LLM report is not displayed when registering a plugin via pytest11.

Key Assertions:

- The `--help` flag is run with `pytester.runpytest` to verify that the LLM report is displayed.
- The `stdout.fnmatch_lines` method checks if the output matches the expected pattern (`*--llm-report*`).
- The plugin registration process completes successfully without any errors or warnings.
- The LLM report is displayed in the console output.
- The `pytester.runpytest` function runs with the correct flags and options.
- The `--help` flag is recognized by pytester.runpytest as a valid option.
- The `stdout` object has a `fnmatch_lines` method that can be used to check the output.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 82ms 7

AI ASSESSMENT

Scenario: Test verifies that special characters in nodeid do not cause the test to crash or produce invalid HTML.

Why Needed: This test prevents a potential regression where special characters in nodeids could cause the Pytest report to fail or be malformed.

Key Assertions:

- The string '
- The HTML content contains only valid HTML tags.
- The report file exists and has a valid HTML structure.

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 a potential bug where the function incorrectly formats less than one minute as '1m Xs', rather than correctly displaying it as 'Xm Xs'.

Key Assertions:

- The result of calling format_duration(60.0) should be exactly '1m 0.0s'.
- The function should handle cases where the input is less than one minute (e.g., 59 seconds) correctly and display it as 'Xm Xs'.
- The function should not incorrectly format more than one minute as 'Xm Xs', but rather as the correct duration string.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	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 where durations are not formatted as expected (e.g., 0.0005 seconds becomes 500 microseconds).

Key Assertions:

- The result contains the string 'µs' to indicate microsecond format.
- The duration is equal to 500 microseconds.
- The function correctly formats a duration of 0.0005 as 500 microseconds.
- The function handles negative durations (e.g., -0.0005) by returning the correct result.
- The function does not silently truncate or round the input values.
- The function preserves the original precision of the input value.
- The function raises an error if the input is not a number.
- The function handles very large durations (e.g., 1.23456789012345678901234567890123456789) 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	2 lines (ranges: 39-40)

PASSED `tests/test_time.py::TestFormatDuration::test_milliseconds_format` 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 500 milliseconds.

Why Needed: Prevents a potential bug where the function does not correctly format durations as milliseconds.

Key Assertions:

- The result should contain 'ms' in its string representation.
- The value of the result should be exactly 500.0 when converted to a float.
- The formatted string should match the expected output, '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: Prevents regression when durations are formatted to include only minutes.

Key Assertions:

- The output should contain 'm' (minutes) and 's' (seconds).
- The output should be in the format '1m 30.5s'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

PASSED tests/test_time.py::TestFormatDuration::test_multiple_minutes 1ms 3

AI ASSESSMENT

Scenario: Tests the 'format_duration' function with a duration of multiple minutes.

Why Needed: Prevents regression due to incorrect handling of multi-minute durations.

Key Assertions:

- The result should be in the format 'mm:ss'.
- The seconds part should be exactly 5.0.
- The minutes part should be exactly 3.
- The function should handle cases where the input is not an integer.
- The function should handle cases where the input is negative.
- The function should handle cases where the input is zero.
- The function should return a string in the 'mm:ss' format for durations greater than one 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/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: Prevents a potential bug where the function incorrectly returns '1s' instead of '1.00s'.

Key Assertions:

- The function should return '1.00s' for a duration of exactly one second.
- The function should handle non-integer inputs correctly.
- The function should not silently truncate decimal places in the output.
- The function should handle very large or very small input values without errors.
- The function should support negative durations.
- The function should return '0.00s' for a duration of exactly zero.
- The function should raise an error if the input is not a non-negative number.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED tests/test_time.py::TestFormatDuration::test_seconds_format 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function to ensure it correctly formats seconds under a minute.

Why Needed: This test prevents regression where the function does not format seconds as 'xx.xxxx' but instead returns 'x.xxxx'.

Key Assertions:

- The result of `format_duration(5.5)` contains the string 's'
- The result of `format_duration(5.5)` is equal to '5.50s'
- The function correctly formats seconds under a 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/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: Test the format of small millisecond durations.

Why Needed: Prevents a potential bug where the function returns incorrect results for very short durations.

Key Assertions:

- The function `format_duration(0.001)` should return '1.0ms' when given a duration of exactly 1 millisecond.
- The function does not raise an error or produce an unexpected result when given a duration that is too small (e.g., negative values, very large positive values).
- The function correctly formats the output as 'X.XXms' for durations between 0.001 and 10 milliseconds.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 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 regression in handling very small durations as microseconds.

Key Assertions:

- The function correctly formats the duration to '1µs' when given a value of 0.000001 seconds.
- The function handles negative values and non-numeric inputs without errors.
- The function preserves the original unit (seconds) in the output.
- The function does not silently truncate or round the result.
- The function correctly handles very small positive and negative values.
- The function raises an error for invalid input types (e.g., strings, lists).
- The function returns a string representation of the duration that matches the expected format.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	2 lines (ranges: 39-40)

PASSED tests/test_time.py::TestIsoFormat::test_formats_datetime_with_utc 1ms 3

AI ASSESSMENT

Scenario: Test the functionality of formatting datetime objects with UTC timezone.

Why Needed: This test prevents a potential bug where the datetime object is not correctly formatted when using the UTC timezone.

Key Assertions:

- The datetime object `dt` is created with the correct timezone (UTC).
- The `iso_format(dt)` method returns the expected string representation of the datetime object in UTC timezone.
- The resulting string does not contain any non-ASCII characters, which is a valid ISO 8601 format for dates and times.
- The time component of the datetime object (`10:30:45`) is correctly formatted as a UTC time zone.
- The AM/PM indicator is correctly handled (e.g., `T` indicates 12-hour clock).
- The resulting string does not contain any invalid characters or formatting errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

PASSED tests/test_time.py::TestIsoFormat::test_formats_naive_datetime 1ms 3

AI ASSESSMENT

Scenario: Tests the ISO format function with a naive datetime without timezone.

Why Needed: Prevents a potential bug where the function incorrectly formats naive datetimes as if they had a timezone.

Key Assertions:

- The output of iso_format(dt) should be '2024-06-20T14:00:00'.
- The format string should match the expected output exactly.
- Any timezone information (like UTC) is ignored in naive datetimes.
- The function handles different date and time formats correctly.
- No exceptions are raised when given a naive datetime without a timezone.
- The function returns the correct result for different input dates and times.
- The function does not perform any unnecessary conversions or calculations.
- Any potential errors in the input data are ignored by the function.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

PASSED `tests/test_time.py::TestIsoFormat::test_formats_with_microseconds` 1ms 3

AI ASSESSMENT

Scenario: Test the `iso_format` function with a datetime object that includes microseconds.

Why Needed: This test prevents a potential bug where the `iso_format` function does not correctly handle datetime objects with microseconds.

Key Assertions:

- The `result` variable should contain the string '123456'.
- The `result` variable should be equal to '123456'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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 current system time is not set to UTC.

Key Assertions:

- The returned datetime object has a valid timezone.
- The returned datetime object's timezone is set to UTC.
- The `tzinfo` attribute of the returned datetime object is not None.
- The `tzinfo` attribute of the returned datetime object is equal to `UTC`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)

PASSED tests/test_time.py::TestUtcNow::test_is_current_time 1ms 3

AI ASSESSMENT

Scenario: Verifies that the `utc_now()` function returns a current time within a specified tolerance when called on UTC.

Why Needed: This test prevents regression in the `utc_now()` function, which may return incorrect results if it is not called on UTC.

Key Assertions:

- The result of `utc_now()` should be within the range `[before, after]` (inclusive).
- The difference between `result` and `after` should be less than or equal to the tolerance specified in the `utc_now()` function.
- The difference between `result` and `before` should be greater than or equal to 0.
- If `utc_now()` is called on a different timezone, it should return an incorrect 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/time.py	1 lines (ranges: 15)

PASSED tests/test_time.py::TestUtcNow::test_returns_datetime 1ms 3

AI ASSESSMENT

Scenario: The function `utc_now()` returns a datetime object.

Why Needed: This test prevents the regression of returning a datetime object when calling `utc_now()`.

Key Assertions:

- result is an instance of `datetime`.
- result has a valid timezone.
- result is not None.
- result is not a string.
- result is not a timedelta.
- result is not a naive datetime object.
- result's tz is set to the UTC timezone.
- result's year, month and day are correctly set.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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,	66, 90-91, 192, 203

					190-191, 194, 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	-