Test Report

Run ID: 20979585481-py3.12 • Generated: 2026-01-14 02:02:28 • Duration: 34.63s
Plugin: v0.1.0 (2f498263985a34902252c53c11fb820445bd8f21) [dirty]
Repo: v0.1.1 (f1a4e8ffe140b3685a0975f68db012dfba4d6df8) [dirty]
LLM: ollama / llama3.2:1b (minimal context, 382 annotated, 4 errors)

92.91%
Total Coverage

387	387	0	0
TOTAL TESTS	PASSED	FAILED	SKIPPED

0	0	0
XFAILED	XPASSED	ERRORS

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

AI ASSESSMENT

Scenario: Test that the aggregate function correctly aggregates all policy when there are multiple test cases.

Why Needed: This test prevents a potential regression where the aggregate function may incorrectly retain only one of the test cases when there are multiple.

Key Assertions:

- The number of tests in the result should be equal to the number of retained test cases.
- Each retained test case should have an outcome of 'passed'.
- All retained test cases should have a duration of 0.1 seconds.
- All retained test cases should belong to the same phase as the first test case.
- The aggregate function should correctly handle multiple test cases without retaining only one.
- The aggregate function should not retain any test cases when there are no matching tests in the input reports.
- The aggregate function should preserve the original order of the test cases within each report.
- Each retained test case should have a unique node ID that matches its parent test case.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the aggregate function fails to work correctly when the input directory does not exist.

Key Assertions:

- aggregator.aggregate() is None
- pathlib.Path.exists(aggregator.aggregate()) is False
- len(aggregator.aggregate()) == 0

COVERAGE

src/pytest_llm_report/aggregation.py	7 lines (ranges: 52, 55-57, 109-111)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the test_aggregate_latest_policy function correctly picks the latest policy for a given test case.

Why Needed: This test prevents regression in cases where two reports with different outcomes are run on the same test at different times.

Key Assertions:

- The outcome of the aggregated report should be 'passed' when the last report has an 'outcome' of 'failed'.
- The number of tests in the aggregated report should be 1.
- The outcome of each individual test in the aggregated report should match the outcome of the latest report.
- The run meta should indicate that this is an aggregated run.
- The run meta should contain a count of 2 runs.
- The summary should have passed and failed counts equal to the number of tests in the aggregated report.

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: Verify that the aggregate function returns None when no directory configuration is provided.

Why Needed: Prevents a potential bug where the aggregate function throws an error or raises an exception without providing a valid directory path.

Key Assertions:

- The aggregate function should return None when `aggregate_dir` is set to `None`.
- The aggregate function should not throw any exceptions or errors when `aggregate_dir` is set to `None`.
- The aggregate function should raise an error with a meaningful message when `aggregate_dir` is set to `None` and the directory does not exist.
- The aggregate function should raise an error with a meaningful message when `aggregate_dir` is set to `None` and the directory path is invalid.
- The aggregate function should not throw any exceptions or errors when `aggregate_dir` is set to `None` and the directory exists.
- The aggregate function should return None when `aggregate_dir` is set to `None` and the directory path is empty.
- The aggregate function should raise an error with a meaningful message when `aggregate_dir` is set to `None` and the directory path is not a valid file system path.

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: Testing the aggregate function when no reports are provided

Why Needed: Prevents a potential bug where an empty report is returned without any errors or warnings

Key Assertions:

- The `aggregate` method should return `None` instead of an empty list when no reports are available.
- An error message should not be printed to the console.
- A warning message should not be printed to the console.
- The function should handle the case where the input is invalid or empty correctly.
- The function should not throw any exceptions when called with a non-empty report.
- The function should return an empty list by default when no reports are provided.

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

Key Assertions:

- coverage: Assert A, Assert B, Assert C were correctly deserialized from JSON.
- llm_annotation: The scenario, why_needed, key_assertions, and confidence of the LLM annotation were correctly set.
- Serialization: The test can be re-serialized with the same 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 that the `aggregate` method returns a single `SourceCoverageEntry` for each source file.

Why Needed: This test prevents regression where multiple source files are aggregated into one entry, leading to incorrect coverage summaries.

Key Assertions:

- The `source_coverage` attribute of the aggregated report should contain exactly one `SourceCoverageEntry`.
- Each `SourceCoverageEntry` in the `source_coverage` list should have a `file_path` attribute matching the source file path.
- All statements, missed, covered, and coverage percentages in each `SourceCoverageEntry` should be accurate and consistent with the original report.
- The `covered_ranges` and `missed_ranges` attributes of each `SourceCoverageEntry` should match the corresponding ranges from the original report.
- The `coverage_percent` attribute of each `SourceCoverageEntry` should be a valid percentage between 0 and 100.
- All statements, missed, covered, and coverage percentages in all `SourceCoverageEntries` should add up to exactly 100% (or match the original report's total coverage).

COVERAGE

src/pytest_llm_report/aggregation.py	66 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 148-155, 157-159, 170, 182, 184-186, 194, 217, 219-220, 235, 245, 248-249, 251, 253, 275-278, 280)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: Prevents regression where the test fails due to an unexpected behavior of the _load_coverage_from_source() method without a configured source file.

Key Assertions:

- The function does not raise any exceptions when the aggregator.config.llm_coverage_source is None.
- The function does not raise any UserWarning when the aggregator.config.llm_coverage_source is '/nonexistent/coverage'.
- The function correctly loads coverage data from a mock .coverage file and returns it as a list of SourceCoverageEntry objects.

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 `test_recalculate_summary` verifies that the aggregator recalculates the latest summary correctly when new test results are added.

Why Needed: This test prevents regression in the aggregator's ability to handle a large number of tests and recalculate the summary accurately.

Key Assertions:

- The total count of tests is updated correctly after adding new test results.
- The passed count remains unchanged despite new failed or skipped tests.
- The failed count increases by one, as expected.
- The skipped count remains unchanged.
- The xfailed count also increases by one, as expected.
- The xpassed count decreases by one, as expected.
- The error count is updated correctly.
- The coverage percentage is preserved and remains at 85.5% after recalculating the summary.
- The total duration of all tests is updated accurately.

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

Why Needed: This test prevents the regression where the aggregator incorrectly counts all reports, including those with missing fields or invalid JSON.

Key Assertions:

- The `aggregate` method should not count any report if it contains missing fields or is marked as invalid.
- The `run_meta.run_count` attribute of the aggregated result should be set to 1 instead of the total number of reports.
- The aggregator should raise a warning when encountering an invalid JSON file, indicating that it's skipping this file.

COVERAGE

src/pytest_llm_report/aggregation.py	71 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119-120, 125, 127-128, 148-153, 155, 157-159, 162, 164-166, 168, 170, 182, 184-186, 194, 217, 219-220, 235, 245, 248-249, 251, 253, 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 given a set of tests with varying durations and coverage totals.

Why Needed: This test prevents regression where the aggregator fails to calculate the summary for tests with different durations or coverage totals.

Key Assertions:

- The total duration of all tests should be equal to the latest summary's total duration.
- At least one test should have a passed status and at least one test should have a failed status.
- The coverage total percent should match the latest summary's coverage total percent.
- The total number of tests should be greater than or equal to 2 (the number of tests provided in the test case).
- All test durations should be non-negative.
- All test coverage totals should be non-negative and add up to at least 100% (the maximum possible coverage total).

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: Testing the caching of tests to prevent skipping them

Why Needed: This test prevents a regression where cached tests are skipped unexpectedly.

Key Assertions:

- The `mock_provider` is not called with any arguments when it's used in the test.
- The `mock_cache` is not set to None after being used in the test.
- The `mock_assembler` is not called with any arguments when it's used in the test.
- The cache is cleared before the next test call, preventing cached tests from being skipped.
- The mock provider returns a valid result for the test.
- The mock cache is set to None after being used in the test.
- The mock assembler does not throw an exception when called with arguments.
- The test logs a message indicating that no cached tests were skipped.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Testing concurrent annotation with multiple providers and caches

Why Needed: Prevents potential performance issues due to concurrent access to shared resources.

Key Assertions:

- Mocking the provider and cache instances ensures they are not accessed simultaneously, preventing race conditions.
- Verifying that the assembler instance is not accessed concurrently allows for efficient annotation processing.
- Ensuring that all mock objects are properly cleaned up after use prevents memory leaks.
- Testing that the annotator function can handle concurrent calls without significant performance degradation.
- Validating that the cache is properly initialized and updated before annotation processing begins.
- Confirming that the assembler instance is properly set up for concurrent annotation execution.
- Checking that the annotator function can handle multiple provider and cache instances concurrently without issues.

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 the correctness of annotated data.

Why Needed: This test prevents a potential regression where concurrent annotations fail to handle failures in the annotator, leading to incorrect results.

Key Assertions:

- The annotator correctly handles failures by re-annotating the data with a new annotation.
- The annotator does not annotate the same data multiple times if it fails.
- The annotator correctly handles failures when the annotator is not available or has an error.
- The annotator logs the failure and continues processing other tasks without interruption.
- The annotator re-annotates the failed data with a new annotation after some time.
- The annotator does not annotate the same data multiple times if it fails within a certain time limit.
- The annotator correctly handles failures when the annotator is running in parallel with other tasks.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the progress reporting function is called with correct arguments.

Why Needed: This test prevents a potential bug where the progress reporting function is not called correctly or at all.

Key Assertions:

- mock_provider is called with the correct mock object
- mock_cache is called with the correct mock object
- mock_assembler is called with the correct mock object

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 with multiple providers and cache.

Why Needed: This test prevents regression in case of concurrent requests or caching issues.

Key Assertions:

- Mocking the provider and cache instance should not affect the annotator's behavior.
- The annotator should be able to handle sequential requests correctly without any issues.
- The cache should not interfere with the annotation process for sequential requests.
- The annotator should return the expected result even when multiple providers are used concurrently.
- The annotator should ignore the cache for sequential requests and focus on the provider's response.
- The annotator should be able to handle different types of providers (e.g., mock, real) without any issues.
- The annotator should not throw any exceptions when dealing with concurrent requests or caching issues.

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 actions when the LLM (Language Model) is disabled.

Why Needed: This test prevents a regression where the annotator would incorrectly skip tests or annotations due to the LLM being disabled.

Key Assertions:

- The `config` object is created with the 'none' provider.
- An empty list `annotate_tests` is passed to the `annotate` function.
- No annotation is performed on any test.
- The annotator does not skip any tests or annotations.
- The LLM is properly disabled without affecting the annotator's behavior.
- The annotator performs its normal workflow when the LLM 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/llm/annotator.py	2 lines (ranges: 45-46)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the annotator skips annotation if the provider is unavailable.

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

Key Assertions:

- Mock Provider Mock
- Skip Annotation
- No Output (no annotation)
- Provider Not Available
- Annotation Skipping
- No Error Message
- No Exception 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/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 when annotating multiple tests concurrently with LLM Max concurrency.

Key Assertions:

- Verify that the `annotate_concurrent` function reports both a successful annotation (2 tasks) and an error (1 task),
- Check if the first error is reported correctly (contains 'first error')
- Ensure that at least one progress message is generated ('Processing 2 test(s)' or 'LLM annotation')
- Verify that the `annotate_concurrent` function does not return all annotations immediately, but instead waits for completion and reports errors 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/llm/annotator.py	28 lines (ranges: 229-232, 234, 236-237, 239-242, 245-246, 248-253, 255-258, 261-264, 266)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: Prevents regression in sequential annotation tasks where the rate limit interval does not have elapsed.

Key Assertions:

- Mocked time.sleep was called with a delay of 0.1s but the expected rate limit interval (1.0s) was not met.
- The mock_time.side_effect was set to [100.0, 100.1, 100.2, 100.3, 100.4] which is slower than the expected rate limit interval of 1.0s.
- The provider.get_rate_limits.return_value was called with None but it should have returned a valid rate limits object.
- The provider.annotate.return_value was called with LlmAnnotation but it should not have been called yet because the rate limit interval has not elapsed.
- The mock_time.side_effect was set to [100.0, 100.1, 100.2, 100.3, 100.4] which is slower than the expected rate limit interval of 1.0s.
- The provider.is_local.return_value was called with False but it should have been True because the task is running on a remote server.
- The provider.get_rate_limits.return_value was called with None but it should have returned a valid rate limits object.
- The provider.annotate.return_value was called with LlmAnnotation but it should not have been called yet because the rate limit interval has not elapsed.
- The mock_time.side_effect was set to [100.0, 100.1, 100.2, 100.3, 100.4] which is slower than the expected rate limit interval of 1.0s.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where cached tests are not reported with their progress.

Key Assertions:

- Verify that any progress messages contain '(cache): test_cached'.
- Check if the 'test_cached' scenario is included in any progress message.
- Ensure that the progress message does not contain '(no cache)'.
- Verify that the progress message contains a string starting with '(cache)'.
- Confirm that the progress message includes the correct context ('src').
- Check if the progress message includes the expected outcome ('passed') for the test_cached scenario.
- Verify that any progress messages are appended to the list 'progress_msgs'.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	37 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-84, 97-98, 100, 127, 129-135, 137, 139)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: To prevent a potential error when the test provider is unavailable, this test verifies that the annotator correctly handles the situation.

Key Assertions:

- mocks.get_provider().is_available() must be called with False as its argument
- mocks.get_provider().is_available() should return False
- mocks.get_provider().get_provider() should throw an exception
- mocks.get_provider().get_provider().should raise a CalledError
- mocks.get_provider().is_available().should not be called again after the first call
- mocks.get_provider().is_available().should return False 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 `test_base_parse_response_malformed_json_after_extract` function checks if the extracted JSON from a malformed JSON string is invalid.

Why Needed: This test prevents a potential bug where the function returns incorrect results when encountering an invalid JSON string after extracting it from a response.

Key Assertions:

- The `annotation.error` attribute should be set to 'Failed to parse LLM response as JSON'.
- The `annotation.json` attribute should be None or empty.
- The `annotation.content` attribute should contain the invalid JSON string.
- The `annotation.type` attribute should be 'Error'.
- The `annotation.message` attribute should contain a clear error message indicating that the response cannot be parsed as JSON.
- The `annotation.errors` attribute should not be empty, if any errors are present.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Tests for non-string fields in response data.

Why Needed: Prevents a potential bug where the base parsing function does not handle non-string fields correctly, potentially leading to incorrect results or errors.

Key Assertions:

- The correct scenario value is '123'.
- The correct why needed list item is ['list'].
- The correct key assertion is 'a'.

COVERAGE

src/pytest_llm_report/collector.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 `gemini` provider configuration.

Key Assertions:

- The function `get_provider(config)` returns an instance of `GeminiProvider`.
- The returned value is not None.
- The returned value is a valid instance of `GeminiProvider`.
- The `provider` attribute of the returned value is set to 'gemini'.
- The `config` parameter passed to `get_provider(config)` has a non-empty string value for the `provider` field.
- The `config` parameter passed to `get_provider(config)` does not contain any invalid values.
- A valid provider configuration can be created using the `Config` class with a valid `provider` argument.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	10 lines (ranges: 52-53, 245, 247, 249, 252, 257, 262-263, 265)
src/pytest_llm_report/llm/gemini.py	7 lines (ranges: 134, 136-139, 141-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that a ValueError is raised when an unknown LLM provider is specified in the configuration.

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

Key Assertions:

- The function `get_provider` raises a `ValueError` with the message 'Unknown LLM provider: invalid'.
- The `pytest.raises()` matcher ensures that the exception type is `ValueError`.
- The `match` parameter specifies the exact error message to match.
- The `invalid` value passed as the `provider` argument should be an unknown LLM provider.
- The `get_provider(config)` function call should raise a `ValueError` with the specified error message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 245, 247, 249, 252, 257, 262, 267)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 bug where the test fails due to an incorrect provider type.

Key Assertions:

- The function `get_litellm_provider` is called with a Config object specifying 'litellm' as the provider.
- The returned value of `get_litellm_provider` is checked to be an instance of LitELLMProvider.
- An assertion error is raised if `get_litellm_provider` returns something other than a LitELLMProvider instance.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where a non-existent provider is used.

Key Assertions:

- The `get_provider` function returns an instance of `NoopProvider` when the `config.provider` attribute is set to `'none'`.
- The `provider` variable holds an instance of `NoopProvider` after calling `get_provider(config)`.
- The `isinstance(provider, NoopProvider)` assertion checks if the returned provider is indeed an instance of `NoopProvider`.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `OllamaProvider` class is not properly instantiated due to missing dependencies.

Key Assertions:

- The `provider` parameter passed to `get_provider(config)` should be an instance of `OllamaProvider`.
- The `get_provider(config)` function should return an instance of `OllamaProvider`.
- The `isinstance(provider, OllamaProvider)` assertion should pass if the returned provider is indeed an instance of `OllamaProvider`.
- The test should fail with a meaningful error message when the `provider` parameter is not an instance of `OllamaProvider`.
- The `get_provider(config)` function should raise a `TypeError` when passed an invalid `provider` type.
- The `config` object passed to `get_provider(config)` should have a valid `provider` attribute.
- The test should only fail if the `provider` parameter is not provided at all.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 test provides a valid LLM provider with available caches.

Why Needed: This test prevents regression in case of an unavailable cache.

Key Assertions:

- The `is_available()` method returns True for both providers.
- The `checks` attribute is incremented by one each time `_check_availability()` is called.
- There are no assertions made on the provider outside of these methods.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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 test verifies that the `get_model_name()` method of a `ConcreteProvider` instance returns the default model name specified in the configuration.

Why Needed: This test prevents a regression where the model name defaults to 'test-model' when no configuration is provided.

Key Assertions:

- assert provider.get_model_name() == 'test-model'
- provider.config.model should be set to 'test-model'
- provider.config.model is not empty
- provider.config.model does not contain any other model names
- provider.config.model is a string
- provider.config.model is a valid Python identifier
- provider.get_model_name() is called with the correct configuration

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 136)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the `get_rate_limits` method of a `ConcreteProvider` instance returns `None` when no rate limits are specified.

Why Needed: This test prevents a potential bug where defaulting to `None` without explicit rate limits in the configuration.

Key Assertions:

- The `get_rate_limits()` method should return `None` for an empty or unspecified configuration.
- The `get_rate_limits()` method should not raise an exception when no rate limits are specified.
- The `get_rate_limits()` method should correctly handle cases where the provider is created with a valid configuration but no rate limits are specified.
- The `get_rate_limits()` method should return `None` for a valid configuration that includes explicit rate limits.
- The `get_rate_limits()` method should not throw an exception when the provider has default rate limits.
- The `get_rate_limits()` method should correctly handle cases where multiple providers are created with different configurations and no rate limits are specified.
- The `get_rate_limits()` method should return `None` for a configuration that includes both explicit and default rate limits.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 128)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: Prevents a regression where `is_local()` returns True when the default is set to true.

Key Assertions:

- The `provider.is_local()` method should return `False` for the default value.
- The `provider.is_local()` method should not be affected by the `local_defaults` configuration option.
- The `provider.is_local()` method should not return True when the `local_defaults` configuration option is set to false.
- The `provider.is_local()` method should raise an error when the default value is set to true and local defaults are disabled.
- The `is_local()` method should be able to distinguish between different configurations (e.g., local vs. non-local)
- The `is_local()` method should not return a cached result for the same configuration
- The `provider.is_local()` method should be able to handle cases where `local_defaults` is set to false but `use_default` is still 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/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 called with the same source code, and it returns the same hash value.

Why Needed: This test prevents a bug where different source codes produce different hashes.

Key Assertions:

- source = "def test_foo(): pass"
- assert hash_source(source) == hash_source(source)
- expected_hash_value = hash_source(source)

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 cache with different source functions.

Why Needed: Prevents a bug where two different source functions could potentially produce the same hash value, leading to unexpected behavior or incorrect results.

Key Assertions:

- The function `hash_source` should return a different hash value for two different source functions.
- Different source functions should not have the same hash value.
- The cache should store and retrieve data correctly even when using different source functions.
- The test should fail if the function `hash_source` is modified to always produce the same hash value.
- Using different source functions in the test case should result in a different output.
- The cache should be able to handle multiple source functions without any issues.
- Using the same source function for both tests should not affect the outcome of the test.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the hash is too short, potentially leading to incorrect caching decisions.

Key Assertions:

- The hash should be at least 16 characters long.
- The hash should not be shorter than 8 characters (assuming a minimum length of 4 for simplicity).
- The hash should not have any leading zeros.
- The hash should not contain any non-alphanumeric characters.
- The hash should not be a palindrome (e.g., 'aabbcc').

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 the `clear` method of LlmCache to ensure it correctly removes all cache entries.

Why Needed: The test prevents a potential bug where some cache entries are not properly cleared when the cache is reset.

Key Assertions:

- Ensure that the clear method returns the correct number of cache entries (2 in this case).
- Verify that the `get` method returns None for both 'test::a' and 'test::b'.
- Check if the cache is properly cleared by verifying its size before and after clearing.

COVERAGE

src/pytest_llm_report/cache.py	26 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 129, 132-136, 141)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Prevents a potential bug where an error-cased annotation would be cached and later retrieved, potentially causing unexpected behavior or errors.

Key Assertions:

- The 'test::foo' cache key should remain empty after setting the annotation.
- The 'test::foo' cache value should not match the original annotation.
- The 'test::foo' cache result should be None.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the function does not return an error or raise an exception when trying to access a non-existent key in the cache.

Key Assertions:

- The function should return `None` when trying to access a non-existent key in the cache.
- The function should throw an exception with a meaningful message when trying to access a non-existent key in the cache.
- The function should raise a KeyError with a meaningful message when trying to access a non-existent key in the cache.
- The function should not return any value (i.e., `None`) when trying to access a non-existent key in the cache.
- The function should throw an exception with a specific error message when trying to access a non-existent key in the cache.
- The function should raise a KeyError with a specific error message when trying to access a non-existent key in the cache.
- The function should not raise any exceptions when trying to access a non-existent key in the cache.

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 from the cache correctly.

Why Needed: Prevents bypass by ensuring annotations are cached before they are used in subsequent requests.

Key Assertions:

- - Check if annotation is set successfully -
- - Check if annotation has the correct confidence level -
- - Verify that the same annotation can be retrieved multiple times with the same key -

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 a collection error has the correct node ID and message.

Why Needed: This test prevents a bug where a collection error is incorrectly identified or reported.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verifies that an empty collection is returned when the `get_collection_errors` method is called on a newly created `TestCollector` instance.

Why Needed: Prevents a potential bug where an empty list is returned when there are no errors in the collection, potentially causing unexpected behavior or errors downstream.

Key Assertions:

- The `get_collection_errors()` method should return an empty list.
- No errors should be present in the collection initially.
- An error message or other indication of collection content should not be present.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents regression when using LLM context override with default value of None.

Key Assertions:

- The llm_context_override attribute of TestCaseResult nodeid='test.py::test_foo' should be None.
- The llm_context_override attribute of TestCaseResult nodeid='test.py::test_foo' is not set to None.
- The llm_context_override attribute of TestCaseResult nodeid='test.py::test_foo' is set to a different value than None.
- The llm_context_override attribute of TestCaseResult nodeid='test.py::test_foo' should be an instance of TestCaseResult.
- The llm_context_override attribute of TestCaseResult nodeid='test.py::test_foo' should not be None.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the default value of llm_opt_out is set to False for a test case with passed outcome.

Why Needed: This test prevents regression where the default value of llm_opt_out might be incorrectly set to True due to a change in the test environment or configuration.

Key Assertions:

- The llm_opt_out attribute of TestCaseResult is checked to be False.
- The llm_opt_out property of TestCaseResult is asserted to be False.
- The value of llm_opt_out is compared with False using the assert method.
- The TestCaseResult object passed into the test_result function has an llm_opt_out attribute that is set to False.
- A TestCaseResult object created with a passed outcome and default llm_opt_out value has its llm_opt_out attribute checked to be False.
- The llm_opt_out property of a TestCaseResult object created with a passed outcome and default llm_opt_out value is asserted to be False.
- The llm_opt_out attribute of a TestCaseResult object created with a passed outcome and default llm_opt_out value is compared with False using the assert 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The `capture` feature is not enabled by default.

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

Key Assertions:

- assert config.capture_failed_output is False
- assert isinstance(config, Config)
- assert 'capture' in config.__dict__
- assert not hasattr(config, '_capture_enabled')
- assert config._capture_enabled is False
- assert config.capture_failed_output == False
- assert config.output_capture_type == '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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- assert config.capture_output_max_chars == 4000
- assert isinstance(config.capture_output_max_chars, int)
- config.capture_output_max_chars >= 1
- config.capture_output_max_chars <= 10000
- config.capture_output_max_chars != 4000

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The `results` dictionary contains the correct key-value pairs for the failed test case.
- The `outcome` attribute of the `result` object matches 'xfailed' according to the expected behavior.
- The `wasxfail` attribute is set to 'expected failure' as expected.
- The `duration` and `longrepr` attributes are correctly initialized with default values.
- The `passed`, `failed`, `skipped` attributes match their expected values for a failed test case.
- The `nodeid` key matches the expected location of the failed test case in the `results` dictionary.

COVERAGE

src/pytest_llm_report/collector.py	36 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-118, 124, 127, 140, 155-159, 163, 167, 171, 209-210, 212, 216, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test 'xfail passes should be recorded as xpassed' verifies that when an xfail is passed, it is correctly marked as xpassed in the test results.

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

Key Assertions:

- The 'when' field of the report should be set to 'call'.
- The 'passed' field of the report should be set to True.
- The 'failed' and 'skipped' fields of the report should both be False.
- The duration of the test run should be 0.01 seconds or less.
- The longrepr field should be an empty string.
- The wasxfail field should be set to 'expected failure'.
- The outcome of the test result should be '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 `create_collector` method of `TestCollector` class.

Why Needed: This test prevents a potential bug where the `TestCollector` instance is created with an empty configuration, leading to incorrect results or behavior.

Key Assertions:

- The `results` attribute of the `collector` object should be an empty dictionary.
- The `collection_errors` list should be an empty list.
- The `collected_count` attribute of the `collector` object should be set to 0.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test 'get_results_sorted' verifies that the collected test results are sorted by node ID.

Why Needed: This test prevents a regression where the order of the test results is not maintained after adding or removing tests.

Key Assertions:

- The list of node IDs in the collected results should be in ascending order.
- The list of node IDs in the collected results should contain only 'a_test.py::test_a' and 'z_test.py::test_z'.
- The node ID 'a_test.py::test_b' is not present in the sorted list.
- The node ID 'b_test.py::test_c' is not present in the sorted list.
- The node IDs are not in ascending order, but rather in descending order.
- There are duplicate node IDs in the collected results.
- The node ID 'z_test.py::test_d' is not present in the sorted list.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify the test collects and deselects items correctly after collection finish.

Why Needed: This test prevents a potential regression where collected items are not deselected after collection finish.

Key Assertions:

- The collector's `collected_count` should be updated to reflect the number of collected items.
- The collector's `deselected_count` should be updated to reflect the number of deselected items.
- The `collected_count` and `deselected_count` attributes should be updated correctly 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 should not capture output if config is disabled and handle_report integration.

Why Needed: To prevent capturing of output when the configuration is set to disable capture of failed outputs via handle_report integration.

Key Assertions:

- ...
- ...
- ...
- collector.handle_runtest_logreport(report) should not be called with a report that has captured stdout.
- result.captured_stdout should be None after collector.handle_runtest_logreport(report).

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the collector captures stderr and reports it correctly.

Why Needed: This test prevents a potential bug where the collector does not capture stderr or reports incorrect stderr messages.

Key Assertions:

- The `captured_stderr` attribute of the `TestCaseResult` object is set to 'Some error'.
- The `report.capstderr` method sets the captured stderr message to 'Some error'.
- The `collector._capture_output(result, report)` function calls `report.capstderr` with the correct value.
- The `result.captured_stderr` attribute is not set to an empty string or None before asserting it.
- The `report.capstdout` method does not have any effect on the captured stderr message.
- The `collector._capture_output(result, report)` function calls `report.capstderr` with a value that matches 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, 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 be set to 'Some output'.
- The `report.capstdout` method should have been called with the correct value ('Some output').
- The `collector._capture_output` function should have recorded the captured stdout 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, 264-265, 268)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the collector truncates output exceeding max chars.

Why Needed: This test prevents a potential bug where the collector does not truncate output exceeding the specified max_chars.

Key Assertions:

- The captured stdout should be truncated to 10 characters.
- The captured stderr is empty.
- The captured stdout contains only 9 characters (1234567890).
- The captured stderr remains unchanged (empty string).

COVERAGE

src/pytest_llm_report/collector.py	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264-265, 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 that the collector extracts item markers correctly when using item markers in the report.

Why Needed: This test prevents a potential regression where the collector might not extract item markers from the report, leading to incorrect results.

Key Assertions:

- The `param_id` attribute of the result is set to 'param1'.
- The `llm_opt_out` attribute of the result is set to True.
- The `llm_context_override` attribute of the result is set to 'complete'.
- All required requirements in the report are extracted and included in the result.
- The item marker with name 'requirement' is correctly retrieved from the collector.
- The item marker with name 'llm_opt_out' is correctly retrieved from the collector.
- The item marker with name 'llm_context_override' is correctly retrieved from the collector.

COVERAGE

`src/pytest_llm_report/collector.py`	35 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 155-159, 163-164, 167-169, 171, 181-182, 185-189, 198-200, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that `collectors.TestCollector._extract_error` correctly handles ReprFileLocation when it causes a crash report.

Why Needed: This test prevents the potential crash of the collector due to an incorrect handling of ReprFileLocation.

Key Assertions:

- The `_extract_error` method should return 'Crash report' when `report.longrepr.__str__.return_value` equals 'Crash report'.
- The `report.longrepr` object's `__str__` method should be called with the argument `'Crash report'`.
- The `report.longrepr` object's `longrepr` attribute should have been set to a string value equal to `'Crash report'`.
- The `_extract_error` method should not raise an exception when `report.longrepr.__str__.return_value` equals '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: The test verifies that the `_extract_error` method of `TestCollector` returns 'Some error occurred' when a `longrepr` is provided.

Why Needed: This test prevents a potential regression where the `longrepr` is not returned correctly if an error occurs during report generation.

Key Assertions:

- The `_extract_error` method of `TestCollector` should return 'Some error occurred' when a `longrepr` is provided.
- The `report.longrepr` attribute should be set to 'Some error occurred'.
- If an error occurs during report generation, the `_extract_error` method should still return '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 there are no longreprs.

Why Needed: Prevents a potential bug where the test fails if there are no longreprs in the report.

Key Assertions:

- The `_extract_skip_reason` method of the `Collector` class should return `None` when `report.longrepr` is `None`.
- The `_extract_skip_reason` method of the `Collector` class should not raise an exception when `report.longrepr` is `None`.
- The `report.longrepr` attribute should be set to `None` before calling `_extract_skip_reason`.
- The `report` object passed to `_extract_skip_reason` should have a `longrepr` attribute.
- The `Collector` class should not throw an exception when called with a `report` object that has no `longrepr` attribute.
- The `_extract_skip_reason` method should be able to extract skip reasons from the report even if there are no longreprs.
- The `Collector` class should correctly handle cases where `report.longrepr` is `None` without raising an exception.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: The test prevents regression when the `longrepr` attribute is not set in the report.

Key Assertions:

- The `report.longrepr` attribute should be an empty string if it's not provided.
- The `report.longrepr` attribute should contain the expected value 'Just skipped' if it's provided.

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 the `extract_skip_reason_tuple` function correctly extracts a skip message from a tuple containing file, line and message information.

Why Needed: Prevents regression where an incorrect or missing reason is extracted for skipped tests.

Key Assertions:

- The `report.longrepr` tuple should contain all three required elements: (file, line, message).
- If the tuple does not contain a `message`, it should still be able to extract the `longrepr` string.
- If the tuple contains an invalid or missing value for any of these elements, the function should raise an error with a descriptive 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::TestCollectorReportHandling::test_handle_collection_report_failure` 1ms 3

AI ASSESSMENT

Scenario: When the `handle_collection_report` method is called with a collection report that fails, it should correctly record the error and update the list of collection errors.

Why Needed: This test prevents a potential bug where the collector does not handle collection reports that fail properly, potentially leading to incorrect or missing error messages.

Key Assertions:

- The `collection_errors` list should contain exactly one element with the specified nodeid and message.
- The first element of `collection_errors` should have the correct nodeid.
- The first element of `collection_errors` should have the correct message.
- The second assertion in `key_assertions` is redundant, as it does not add any new information to the test.

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: The test verifies that the `handle_runtest_rerun` method correctly handles reruns by setting the `rerun` attribute of the report to 1 and asserting its count is 1, while also ensuring the final outcome is 'failed'.

Why Needed: This test prevents a regression where the `rerun` attribute is not being set correctly after a rerun.

Key Assertions:

- res.rerun_count == 1
- assert res.final_outcome == "failed"
- res.rerun == 1

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents regression by ensuring that the TestCollector correctly handles setup errors and logs them properly.

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: TestCollectorReportHandling::test_handle_runtest_teardown_failure verifies that the test collects an error when teardown fails after a pass.

Why Needed: This test prevents a regression where the test might not collect errors in cases of teardown failures, potentially masking issues with cleanup.

Key Assertions:

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

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 'Gemini' booster is not correctly handling edge cases where no preferred models are specified or when all models are specified.

Key Assertions:

- The function provider._parse_preferred_models() returns a list of strings containing 'm1', 'm2'.
- The function provider._parse_preferred_models() returns an empty list when the model is None.
- The function provider._parse_preferred_models() returns an empty list when the model is set to 'All'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	17 lines (ranges: 134, 136-139, 141-142, 385, 387, 417-424)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the rate limiter correctly handles edge cases where there are more tokens available than requests.

Why Needed: This test prevents a bug where the rate limiter allows for excessive token usage without triggering an error.

Key Assertions:

- assert limiter.next_available_in(60) > 0
- assert limiter.next_available_in(10) == 0
- assert limiter.tokens() < 150
- assert limiter.request_count() < 50
- assert len(limiter.history()) == 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/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: Test the `to_dict()` method of SourceCoverageEntry and LlmAnnotation classes to ensure they return expected values for coverage percent, error messages, and run metadata.

Why Needed: This test prevents regressions where coverage percent or error message is not correctly returned when using these classes.

Key Assertions:

- The `coverage_percent` attribute of SourceCoverageEntry should be equal to the provided value (50.0).
- The `error` attribute of LlmAnnotation should be equal to the provided value ('timeout').
- The `duration` attribute of RunMeta should be equal to the provided value (1.0).

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: To prevent a potential bug where the mapper's warnings are not properly initialized or populated.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents bugs or regressions where the `get_warnings` method returns an incorrect type (e.g., a non-list), potentially causing issues downstream in the application.

Key Assertions:

- The `warnings` variable is expected to be of type `list`.
- The `warnings` list contains at least one warning.
- Each warning in the `warnings` list has the correct structure (i.e., it's a dictionary with keys 'message' and 'code').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	3 lines (ranges: 44-45, 308)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that `map_coverage` returns an empty dictionary when no coverage file exists.

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

Key Assertions:

- The function should return an empty dictionary.
- The `Path.exists` mock should return False.
- The `glob.glob` mock should return an empty list.
- The `map_coverage` function should not raise any exceptions.
- The test should have at least one warning.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Should include all phases when `include_phase=all` configuration is used.

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

Key Assertions:

- The function `_extract_nodeid` should return the correct node ID for each phase.
- The function `_extract_nodeid` should handle cases where the phase name contains a dot (.) correctly.
- The function `_extract_nodeid` should include all phases in the coverage map even when `include_phase=all` is specified.

COVERAGE

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

Why Needed: This test prevents a potential bug where the `CoverageMapper` does not extract node IDs from setup phases, potentially leading to incorrect coverage reports.

Key Assertions:

- The function `_extract_nodeid` is called with the correct arguments (`''test.py::test_foo|setup''`) and returns `None` as expected.
- The test asserts that the extracted node ID is `None` for the specified module and phase.
- The `CoverageMapper` instance has a valid configuration object with the `include_phase` set to `''run''`.
- The `nodeid` variable is assigned the correct value after calling `_extract_nodeid` on the `mapper` 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	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: The test verifies that the `test_extract_nodeid_with_run_phase` function correctly extracts a node ID from the run phase context.

Why Needed: This test prevents bugs or regressions where the coverage map might not accurately reflect the code's execution phases.

Key Assertions:

- The extracted node ID should match the expected value `test.py::test_foo`.
- The function `_extract_nodeid` is correctly called with the correct arguments `('test.py::test_foo|run')`.
- The coverage map is updated to reflect the accurate execution phase of the code being tested.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the `extract_contexts` method of `CoverageMapper` correctly extracts all contexts in a given file.

Why Needed: This test prevents regression where the coverage map does not include all paths in the `_extract_contexts` method, potentially leading to incorrect analysis or missed context areas.

Key Assertions:

- The function should return a list of covered lines for each context.
- The function should return an empty list for files that do not have any contexts.
- Each line within a covered context should be counted correctly and in the correct order.
- The function should handle cases where multiple files are covered by the same context.
- The function should ignore non-python files when extracting contexts.
- The function should return an empty list for files that do not have any contexts, even if they contain code.
- The function should correctly handle cases where a file has multiple lines with the same line number.
- The function should preserve the original order of covered lines within each context.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	57 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 137-140, 144, 148, 150, 152-153, 156, 160-163, 165, 167-168, 173, 176, 178-184, 187-189, 191-194, 196, 199-200, 202, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	17 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64, 100, 103, 111-112, 116, 123)
src/pytest_llm_report/util/ranges.py	13 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65-67)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the method incorrectly assumes all files have test contexts and returns an incorrect result.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the `CoverageMapper` with different node IDs and phases.

Why Needed: This test prevents regression in coverage analysis when missing lines or without specific phases.

Key Assertions:

- The `_extract_nodeid()` method should return the expected node ID for each phase.
- The `_extract_nodeid()` method should return `None` for nodes with no specified phase.
- The context without a pipe character (`|`) should be correctly identified as having no phase.
- The test should pass even when the input string contains only one node ID (e.g., `test.py::test`).
- The test should fail when the input string contains multiple nodes with different phases (e.g., `test.py::test|setup` and `test.py::test|run`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	14 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231-234, 236, 239)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test 'test_load_coverage_data_no_files' verifies that the test fails when no coverage files exist.

Why Needed: This test prevents a regression where the test would silently fail or produce incorrect results when there are no coverage files.

Key Assertions:

- The function `_load_coverage_data()` returns `None` when no .coverage files exist.
- The warning message is 'W001' as expected.
- There is only one warning raised in this test case.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the `CoverageMapper` raises an error when trying to load corrupt coverage files.

Why Needed: This test prevents a potential regression where the `CoverageMapper` fails to handle corrupted coverage data, potentially leading to unexpected behavior or errors in downstream applications.

Key Assertions:

- The function `_load_coverage_data()` should raise an exception with the message 'Corrupt coverage file' when trying to read from a mock CoverageData instance that raises an error on read.
- Any warnings generated by the `CoverageMapper` should include the string 'Failed to read coverage data'
- The function `mapper.warnings` should contain at least one warning object with the message 'Corrupt coverage file'
- The function `mapper.warnings` should not contain any other error messages
- The function `_load_coverage_data()` should return None
- Any exceptions raised by the mock CoverageData instance should be caught and 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/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 the correct number of updates.

Why Needed: This test prevents regression in handling parallel coverage files, which is crucial for ensuring accurate coverage data.

Key Assertions:

- The `update` method of the `CoverageData` class was called twice with different mock instances.
- The number of calls to `update` should be greater than or equal to 2.
- The mock instances returned by `mock_data_cls.side_effect` should not have been garbage collected before being used in `_ = mapper._load_coverage_data()`
- The coverage data loaded from the parallel files should contain both mock instances.
- The number of unique coverage data objects (i.e., different instances) should be greater than or equal to 2.
- The `update` method of the `CoverageData` class was called with at least one mock instance.
- The `update` method of the `CoverageData` class was called without any mock instances being garbage collected

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	15 lines (ranges: 44-45, 72-73, 83, 86, 88, 92, 94, 98, 101-104, 106)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test the `map_coverage` method when it returns an empty dictionary.

Why Needed: Prevents a potential bug where the test fails with an incorrect assertion when no coverage data is available.

Key Assertions:

- The `_load_coverage_data` method should return None or an empty dictionary when called without any arguments.
- The `map_coverage` method should correctly handle this case and return an empty dictionary.
- The test should pass even if the `_load_coverage_data` method returns a non-empty dictionary, as it is not relevant to the expected behavior of the `map_coverage` 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/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 CoverageMapper handles analysis2 errors during source coverage analysis.

Why Needed: To prevent unexpected behavior when analyzing code with errors, such as skipping files with errors.

Key Assertions:

- The `map_source_coverage` method should return an empty list of entries for mock_cov.analysis2.side_effect == Exception('Analysis failed')
- The `map_source_coverage` method should not skip any files in the case of analysis2 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: Verify that the test maps all source files in `map_source_coverage` to their corresponding coverage statistics.

Why Needed: This test prevents a regression where some source files are not covered by the analysis, leading to incomplete coverage reports.

Key Assertions:

- The function `map_source_coverage` returns exactly one entry with file path `app.py` and its corresponding coverage statistics.
- The first assertion checks that the number of entries returned is equal to 1.
- The second assertion verifies that the file path matches `app.py`.
- The third assertion ensures that the statements in the entry are equal to 3.
- The fourth assertion confirms that the covered percentage is equal to 66.67.
- The fifth assertion checks that there are no missing entries with a coverage percentage greater than 0.
- The sixth assertion verifies that the coverage percentage of the first entry matches 66.67.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: To prevent a potential bug where an unknown warning is returned instead of a specific one, such as W001_NO_COVERAGE.

Key Assertions:

- The function `make_warning` should return an instance of class WarningCode.W001_NO_COVERAGE with the correct code.
- The message of the returned warning should contain 'No .coverage file found'.
- The detail of the returned warning should be 'test-detail'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that warning codes have correct values.

Why Needed: Prevents a potential bug where the wrong warning code is assigned to an error condition.

Key Assertions:

- {'message': 'Assertion failed: WarningCode.W001_NO_COVERAGE.value == "W001"'}
- {'message': 'Assertion failed: WarningCode.W101_LLM_ENABLED.value == "W101"'}
- {'message': 'Assertion failed: WarningCode.W201_OUTPUT_PATH_INVALID.value == "W201"'}
- {'message': 'Assertion failed: WarningCode.W301_INVALID_CONFIG.value == "W301"'}
- {'message': 'Assertion passed: WarningCode.W401_AGGREGATE_DIR_MISSING.value == "W401"'}

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevent a warning that occurs when trying to convert a `Warning` object to a dictionary without setting 'detail' attribute.

Key Assertions:

- The `code` attribute is set correctly and matches the expected value.
- The `message` attribute is set correctly and matches the expected value.
- The `detail` attribute is not set, which should prevent this warning from occurring.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test verifies that the `make_warning` function creates a warning with the correct code and message.

Why Needed: This test prevents a potential bug where the warning is not created correctly when the code is known to be valid.

Key Assertions:

- The warning has the expected code `WarningCode.W101_LLM_ENABLED`.
- The warning has the expected message from `WARNING_MESSAGES` dictionary.
- The warning detail is None, indicating that no additional information 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/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 code is not covered by warnings, leading to unexpected behavior.

Key Assertions:

- The function `make_warning` does not throw an exception when given an unknown warning code.
- The function `make_warning` uses the fallback message 'Unknown warning.' for unknown warning codes.
- The function `make_warning` restores the original message after using it to make a warning.
- The function `make_warning` correctly handles missing warning messages in the dictionary.
- The function `make_warning` preserves the original message when restoring 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/errors.py`	4 lines (ranges: 139-142)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test 'test_make_warning_with_detail' verifies creating a warning with detailed message.

Why Needed: This test prevents the creation of warnings without any detail, which can lead to unexpected behavior or errors in the application.

Key Assertions:

- w.code == WarningCode.W301_INVALID_CONFIG
- w.detail == 'Bad value'
- assert w.detail is not None
- assert isinstance(w.detail, str)
- assert len(w.detail) > 0
- assert w.detail != '',
- assert w.detail != 'Bad value'

COVERAGE

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

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

AI ASSESSMENT

Scenario: Ensures that all enum values are strings and start with 'W', preventing potential warnings.

Why Needed: This test prevents WarningsCodes from being converted to integers without proper string representation.

Key Assertions:

- code.value should be a string.
- code.value should start with 'W'.
- All enum values must have a string representation.
- WarningCode enum values cannot be converted to integers without explicit conversion.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the warning to dictionary serialization without detail.

Why Needed: Prevents a potential bug where warnings are not properly serialized to dictionaries, potentially leading to incorrect data being stored in test results.

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 that WarningDataClass can be serialized correctly with detail.

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

Key Assertions:

- The 'code' key in the dictionary should match the actual code value.
- The 'message' key in the dictionary should match the actual message value.
- The 'detail' key in the dictionary should match the actual detail value.
- The 'WarningCode.W001_NO_COVERAGE' key should be present in the dictionary and have the correct value.
- The 'Check setup' key should also be present in the dictionary with the correct value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Test verifies whether a given file extension is 'python' or not.

Why Needed: This test prevents potential issues where the `is_python_file` function incorrectly identifies non-.py files as Python files.

Key Assertions:

- The function should return False for files with extensions other than '.py'.
- The function should return False for files with extensions like 'foo/bar.txt' or 'foo/bar.pyc'.
- The function should not incorrectly identify 'foo/bar.py' as a non-.py file.
- The function should handle case-insensitive matching of file extensions.
- The function should correctly handle files with multiple '.' characters in their extension.
- The function should return False for files without any extension (e.g., just 'foo.txt').
- The function should raise an error when given a non-existent Python file (e.g., 'non_existent.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 returns True for a `.py` file.

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

Key Assertions:

- The function should return `True` when given a string representing a `.py` file.
- The function should raise an error or return a specific value indicating that it cannot identify the file type.
- The function should handle cases where the input is not a valid Python file (e.g., a non-`.py` file with Python code).
- The function should correctly handle nested directories and relative paths in the file path.
- The function should ignore case when comparing file extensions (e.g., `.Py` vs. `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 by creating a subdirectory and verifying the resulting file path is correct.

Why Needed: This test prevents a potential issue where the `make_relative` function does not correctly handle cases where the input file path has an absolute component.

Key Assertions:

- The `make_relative` function should create a new file path that includes the subdirectory specified in the first argument and the original file name.
- The resulting file path should be relative to the base directory of the test.
- The parent directory of the input file path should not be included in the output file path.
- The `touch` method should create a new file with the correct permissions (in this case, no modifications).
- The `mkdir` method should create the subdirectory if it does not already exist.
- The `exist_ok` parameter should prevent an error from being raised if the parent directory already exists.
- The resulting file path should have a relative extension ('.py').
- The original file name should be included in the output file path (e.g., 'file.py').

COVERAGE

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

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

AI ASSESSMENT

Scenario: tests/test_fs.py::TestMakeRelative::test_returns_normalized_with_no_base

Why Needed: Prevents a potential bug where the function returns an incorrect normalized path when no base is provided.

Key Assertions:

- The result of `make_relative('foo/bar')` should be 'foo/bar'.
- The current implementation does not handle cases where the input directory contains only subdirectories.
- A normalization step would be required to ensure consistency across different operating systems and file systems.
- Without this check, the function could return an incorrect normalized path in certain edge cases.
- This test verifies that the `make_relative` function behaves correctly when no base is provided.
- The test ensures that the function returns a normalized path as expected.
- It also checks if the current implementation handles subdirectories correctly.
- In case of subdirectories, the function should still return a 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	7 lines (ranges: 30, 33, 36, 39, 42, 55-56)

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

AI ASSESSMENT

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

Why Needed: This test prevents potential issues where the `normalize_path` function may incorrectly handle already-normalized paths, potentially leading to unexpected behavior or errors in downstream applications.

Key Assertions:

- Input: 'foo/bar'
- Output: 'foo/bar'
- Path is already normalized
- Normalization does not affect the path's readability or usability

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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 that the `normalize_path` function correctly converts forward slashes in file paths.

Why Needed: This test prevents a potential bug where the function incorrectly handles forward slashes in file paths.

Key Assertions:

- The input string should be converted to 'foo/bar' without any changes.
- The resulting path should have the same structure as the original input.
- Any backslashes in the input string should be replaced with forward slashes.
- The function should not add any additional slashes to the end of the path.
- The function should handle paths with multiple consecutive forward slashes correctly.
- The function should ignore leading or trailing whitespace characters in the input string.
- The function should preserve the original directory structure when converting backslashes to forward slashes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

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

AI ASSESSMENT

Scenario: Tests the `normalize_path` function to remove trailing slashes from paths.

Why Needed: Prevents a potential bug where a path with a trailing slash is returned unmodified.

Key Assertions:

- The input path should be stripped of any trailing slashes.
- The resulting normalized path should have no leading or trailing slashes.
- Any empty string should not be returned as the normalized path.
- A path with only one slash should remain unchanged.
- A path with multiple consecutive slashes should also remain unchanged.
- A path with a single slash followed by another slash should be considered as having only one slash.
- The function should handle paths with leading or trailing whitespace correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

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

AI ASSESSMENT

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

Why Needed: Prevents potential issues where paths are incorrectly excluded due to custom patterns.

Key Assertions:

- The 'should_skip_path' function should return True when the path matches any of the custom exclusion patterns.
- The 'should_skip_path' function should return False when the path does not match any of the custom exclusion patterns.
- The 'exclude_patterns' parameter should be able to exclude specific file extensions or patterns from being skipped.
- Custom exclusion patterns should be able to override the default behavior of the 'should_skip_path' function.
- The test should cover both cases where a path matches and does not match any custom exclusion patterns.
- The test should use the correct syntax for passing custom exclusion patterns as arguments to the 'exclude_patterns' parameter.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	15 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116-117, 119-121, 123)

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

AI ASSESSMENT

Scenario: tests/test_fs.py::TestShouldSkipPath::test_normal_path

Why Needed: To prevent skipping of normal paths due to incorrect implementation of 'should_skip_path' function.

Key Assertions:

- The 'should_skip_path' function should return False for the path 'src/module.py'.
- The 'should_skip_path' function should not throw an exception or raise an error when given a valid path.
- The 'should_skip_path' function should maintain its original behavior for other paths (e.g., directories, files).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116, 123)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the function incorrectly considers non-`.git` directories as paths to be skipped.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the `should_skip_path` function incorrectly includes `__pycache__` directories in its skip list, leading to unexpected behavior or errors.

Key Assertions:

- The file path should be relative and start with `__pycache__`.
- The file path should not contain a `.pyc` extension.
- The file path should be located within the `foo/__pycache__` directory.
- The file path should not be located at the root of the test directory.
- The file path should be skipped by the `should_skip_path` function.
- The file path should not cause any errors or unexpected behavior when passed to the `should_skip_path` function.

COVERAGE

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

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

AI ASSESSMENT

Scenario: tests/test_fs.py::TestShouldSkipPath::test_skips_venv

Why Needed: This test prevents a potential issue where the `should_skip_path` function incorrectly identifies venv directories as being to be skipped.

Key Assertions:

- The function should return True for venv directories, but it currently returns False.
- The function should return True for .venv directories, but it currently returns False.
- The function should handle the case where a directory contains multiple venv subdirectories correctly.
- The function should not incorrectly identify a non-existent venv directory as being to be skipped.
- The function should raise an exception when given a non-venv directory path instead of returning False.
- The function should ignore the presence of a parent directory containing a venv subdirectory in its own case.
- The function should handle the case where a Python package is installed in a virtual environment but not in the current working directory correctly.
- The function should raise an exception when given a non-venv directory path with a non-existent parent directory.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that pruning clears request and token usage times when a request is added in the past

Why Needed: This test prevents regression where adding a request in the past would cause pruning to clear both request and token usage times.

Key Assertions:

- _request_times should be empty after pruning
- _token_usage should be empty after pruning
- limiter._prune(time.monotonic()) should not modify _request_times or _token_usage
- assert len(limiter._request_times) == 0 and assert len(limiter._token_usage) == 0 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 when the limit is set to 1 request per minute.

Why Needed: This test prevents a potential issue where a client can send multiple requests in quick succession, potentially overwhelming the server and causing it to become unavailable.

Key Assertions:

- The `next_available_in` method returns a value greater than 0 when the limit is set to 1 request per minute.
- The `next_available_in` method returns a value less than or equal to 60.0 seconds when the limit is set to 1 request per minute.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test verifies that the rate limiter does not allow tokens to be used before they become available, preventing potential abuse and ensuring fair usage of the API.

Key Assertions:

- The next_available_in method returns a non-zero value (greater than 0) when there are tokens left.
- The _token_usage list is updated correctly after recording tokens.
- The number of tokens used in the last 20 seconds is less than or 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/llm/gemini.py	33 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 81-82, 84, 87-88, 92-94, 100-101, 103, 105, 107-108, 110-114)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- limiter.wait_for_slot() was called with mock_sleep
- the `time.sleep` function was called exactly once
- no other requests were made during the test duration

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	31 lines (ranges: 39-42, 45-46, 48, 52-54, 58-59, 61-63, 73, 76-78, 81-82, 84, 87-88, 92-93, 95, 97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the limiter does not record tokens for the first few minutes of usage.

Key Assertions:

- The length of `_token_usage` is equal to 0 after calling `record_tokens(0)`.
- The number of records in `_token_usage` is exactly 1 (i.e., zero tokens were recorded).
- The limiter's token usage does not exceed the specified limit (100 tokens per minute) for at least 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/llm/gemini.py	6 lines (ranges: 39-42, 66-67)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify the test Gemini Limiter requests per day exhaustion.

Why Needed: This test prevents a potential rate limit exceeded error when exceeding daily request limits.

Key Assertions:

- The `GeminiRateLimitExceeded` exception is raised with the correct message 'requests_per_day'.
- The `wait_for_slot` method returns an instance of `_GeminiRateLimitExceeded` exception.
- The `record_request` method does not raise a `GeminiRateLimitExceeded` exception when exceeding daily limits.
- The `wait_for_slot` method waits for the requested slot to become available, allowing for requests beyond the limit.
- The `requests_per_day` attribute is set to 1, indicating a single request per day.
- The test does not verify that the `GeminiRateLimitExceeded` exception is raised with an error message related to 'requests_per_day'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	24 lines (ranges: 32-34, 39-42, 45-46, 48-50, 58-60, 73, 76-78, 81-82, 84, 87-88)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test the tpm wait time fallback feature of GeminiRateLimiter.

Why Needed: This test prevents a potential regression where the rate limiter fails to detect and prevent excessive TPM usage without sufficient tokens.

Key Assertions:

- The current rate limit is not being exceeded by the request tokens.
- The token usage is not empty after filling up the TPM.
- The time until TPM becomes available is greater than 0 seconds.
- _seconds_until_tpm_available returns a positive 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 556ms 6

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the RPM rate limit cooldown might not be properly handled, leading to unexpected behavior or errors in subsequent tests.

Key Assertions:

- The 'models/gemini-pro' model should be present in the provider's cooldowns.
- The cooldown time for the 'models/gemini-pro' model should be greater than 1000.0 seconds (1 minute).
- The provider's cooldowns dictionary should contain the 'models/gemini-pro' 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	23 lines (ranges: 52-53, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	117 lines (ranges: 32-34, 39-42, 45-46, 48, 52-54, 66, 68-70, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-215, 217-223, 225-227, 233-234, 238-240, 242-243, 274-277, 280, 282-290, 292-295, 297-298, 300-301, 346, 348-350, 352-353, 381-382, 385-386)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- The annotation should contain the correct scenario 'Recovered Scenario'
- The mock_post call count should be equal to 2
- The annotation should not have an error message
- The annotation should contain a valid model name
- The annotation should contain supported generation methods
- The annotation should contain a valid content snippet
- The annotation should contain the correct part of the text

COVERAGE

src/pytest_llm_report/collector.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 the correct annotation when a successful response is received from _call_gemini.

Why Needed: This test prevents regression in case of an unexpected error or incorrect response from _parse_response.

Key Assertions:

- The annotation returned by _annotate_internal has the expected scenario and no error.
- The annotation does not contain any error information.
- The annotation's `scenario` field matches the expected value.
- _parse_response returns a Mock object with the correct scenario and no error.
- The annotation's `error` field is None.
- The annotation's `text` field contains the expected text.
- The annotation's `tokens` field does not contain any tokens.
- The annotation's `_call_gemini` method returns the correct 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	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: Verify that the GeminiProvider class checks availability correctly when no environment variables are set.

Why Needed: This test prevents a potential bug where the provider's availability check fails due to missing environment variables, potentially leading to incorrect API token usage or other issues.

Key Assertions:

- The provider._check_availability() method returns False if no environment variables are set.
- The provider._check_availability() method should return True when environment variables are set with a valid API token.
- The provider._check_availability() method should raise an AssertionError or other exception when environment variables are not set and the API token is invalid or missing.
- The provider._check_availability() method should not throw an AssertionError if environment variables are set but the API token is invalid or missing.
- The provider._check_availability() method should not return True if environment variables are set with a valid API token, but the API token is invalid or missing.
- The provider._check_availability() method should raise an AssertionError when environment variables are set with a valid API token, but the API token is invalid or missing.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	10 lines (ranges: 134, 136-139, 141-142, 266-267, 269)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents excessive requests within a certain time frame.

Why Needed: This test prevents a potential infinite loop of requests due to the rate limiter not being able to handle high request rates.

Key Assertions:

- The next_available_in() method should return None after 100 requests.
- The limiter's record_request() method should be called before calling next_available_in().
- The limiter's next_available_in() method should not be called within the same request.
- The rate limit should not be exceeded for any given time period (in this case, 100 requests).
- The limiter's record_request() method should increment the request count and then call next_available_in().
- The limiter's next_available_in() method should return None after calling record_request().
- The rate limit should not be exceeded within a certain time frame (in this case, 100 requests).

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the rate limiter does not block subsequent requests when there are no available slots in the current window.

Why Needed: This test prevents a potential bug where subsequent requests to the same endpoint would block due to insufficient available slots in the rate limit window.

Key Assertions:

- The next_available_in function should return 0.0 after three requests without any available slots in the rate limit window.
- The limiter.record_request() function should not be called before the first request has a valid available slot in the rate limit window.
- The wait value returned by next_available_in should be greater than 0 and less than or equal to 60.0 after three requests without any available slots in the rate limit window.

COVERAGE

src/pytest_llm_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 yield different hashes.

Why Needed: Prevents regression in case of provider switching, where the hash may change.

Key Assertions:

- The function `compute_config_hash` should return a different hash for two instances with different config providers.
- The function `compute_config_hash` should not return the same hash when both providers are 'none'.
- The function `compute_config_hash` should not return the same hash when one provider is 'ollama' and the other is 'none'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verifies that the computed hash is short and returns a 16-character string.

Why Needed: Prevents potential security vulnerabilities by ensuring the hash length is sufficient to prevent collisions.

Key Assertions:

- The length of the computed hash should be exactly 16 characters.
- The hash should not contain any non-ASCII characters.
- The first character of the hash should be a lowercase letter.
- The second character of the hash should be a lowercase letter.
- The third character of the hash should be a digit.
- The fourth character of the hash should be an uppercase letter.
- The fifth character of the hash should be a digit.
- All characters in the hash should be either lowercase letters or digits.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a bug where the computed SHA-256 hash does not match the content hash even if the input file is the same.

Key Assertions:

- The computed SHA-256 hash of the file should be equal to its content hash.
- The content hash of the file should be equal to the computed SHA-256 hash.
- The computed SHA-256 hash of a different file with the same input should also match the content hash.
- If the input file is modified, the computed SHA-256 hash should still match the content hash.
- If the input file is deleted or renamed, the computed SHA-256 hash should still match the content hash.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	6 lines (ranges: 32, 44-48)

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

AI ASSESSMENT

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

Why Needed: Prevents potential issues where incorrect or incomplete file contents are passed to `compute_file_sha256` function, potentially leading to incorrect hashes.

Key Assertions:

- The length of the computed hash should be exactly 64 bytes.
- The computed hash should not contain any zeros (indicating an empty string).
- Any non-zero values in the file contents should be present in the hash output.
- No leading zeros in the hash output are expected, as they indicate a truncated hash.
- The hash is not empty and does not start with '0x' prefix.
- The hash contains only hexadecimal digits (a-z, A-Z, 0-9).
- Any non-hexadecimal characters in the file contents should be ignored during hashing.

COVERAGE

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

Why Needed: Prevents a potential issue where two different keys may generate the same signature, which could be exploited by an attacker to bypass security measures.

Key Assertions:

- The computed signature for key1 is different from the computed signature for key2.
- The computed signature for key1 is not equal to the expected value (which should be different due to the different keys).
- The computed signature for key2 is different from the computed signature for key1.
- The computed signature for key2 is not equal to the expected value (which should be different due to the different keys).

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verifies the computation of HMAC using a secret key.

Why Needed: Prevents a potential issue where an attacker could exploit the lack of padding or authentication in the HMAC calculation.

Key Assertions:

- The length of the generated signature is 64 bytes.
- The signature is not empty.
- The signature contains only hexadecimal digits.
- No leading zeros are present in the signature.
- No trailing zeros are present in the signature.
- The signature does not contain any whitespace characters.
- The signature does not contain any newline characters.
- The signature does not contain any special characters other than those allowed by the HMAC algorithm.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The 'compute_sha256' function is called with the same input and produces the same output.

Why Needed: This test prevents a bug where different inputs to the `compute_sha256` function produce different hashes, potentially leading to incorrect results or security vulnerabilities.

Key Assertions:

- The two hash values should be equal.
- The contents of the input bytes should not change the output hash.
- The hash value should remain constant for a given set of input data.
- Different inputs to `compute_sha256` function should produce different hashes.
- The hash value should not be affected by changes in the input data, such as padding or encoding.
- The hash value should not be changed by appending or modifying the input bytes.
- The hash value should remain unchanged if the input is a known constant (e.g., an empty string).
- The hash value should not change when the input is modified in-place (e.g., using `b'...' = b'...'`) but the function is called again.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 32)

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

AI ASSESSMENT

Scenario: The hash of the input string 'test' is expected to have a length of 64 characters.

Why Needed: This test prevents a potential SHA-256 hashing issue where the output may not be exactly 64 bytes due to various factors such as encoding or padding errors.

Key Assertions:

- The hash should be a string of 64 hexadecimal characters.
- The length of the hash should be exactly 64 characters.
- The hash should have no leading zeros.
- The hash should not contain any null bytes.
- The hash should only contain hexadecimal digits (0-9, A-F, a-f).
- The hash should not contain whitespace or special characters.
- The hash should be a valid SHA-256 hash.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 32)

PASSED `tests/test_hashing.py::TestGetDependencySnapshot::test_includes_pytest` 81ms 3

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the 'pytest' package is not included in the dependency snapshot.

Key Assertions:

- The 'pytest' package should be present in the dependency snapshot.
- The 'pytest' package should be listed as an item in the snapshot.
- The snapshot should include all required dependencies, including 'pytest'.
- The test should fail if the 'pytest' package is not included in the dependency snapshot.
- The test should fail if the 'pytest' package is missing from the snapshot.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 113-114, 116-121)

PASSED tests/test_hashing.py::TestGetDependencySnapshot::test_returns_dict 82ms 3

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the function may not return a dictionary if it encounters an error or edge case.

Key Assertions:

- snapshot is an instance of dict
- snapshot has no attributes other than __dict__
- the `isinstance` check passes with `snapshot` as dict

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
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: The test verifies that the `load_hmac_key` function correctly loads a key from a file.

Why Needed: This test prevents a bug where the HMAC key is not loaded correctly from the file, potentially leading to incorrect authentication.

Key Assertions:

- A bytes object containing the expected HMAC key should be returned by the `load_hmac_key` function.
- The HMAC key in the file should match the expected value.
- The `load_hmac_key` function should raise an error if the file is not found or cannot be read.
- The `load_hmac_key` function should correctly decode the HMAC key from the file bytes.
- The `load_hmac_key` function should handle cases where the file is empty or contains only whitespace.
- The `load_hmac_key` function should raise an error if the file does not contain a valid HMAC key.
- The `load_hmac_key` function should correctly load the HMAC key from the first line of the file (assuming it's the only one).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: To prevent a potential bug where the test fails due to an unexpected exception being raised when trying to load a non-existent key file.

Key Assertions:

- The `load_hmac_key` function should be able to successfully load the HMAC key from the provided configuration.
- The `load_hmac_key` function should not raise any exceptions if the key file does not exist.
- The test should fail with a clear error message indicating that the key file is missing.
- The test should verify that the expected exception (KeyFileNotFoundError) is raised when trying to load a non-existent key file.
- The `Config` class should be able to handle the case where the key file does not exist without raising an exception.
- The test should only fail if the key file exists, and not if it's just missing for some reason (e.g. due to permissions issues).
- The test should provide a clear indication that the key file is missing by verifying its absence in the configuration object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	4 lines (ranges: 73, 76-78)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `load_hmac_key` function should return `None` if no key file is provided.
- No exception should be raised when no key file is specified.
- The function should correctly identify and return `None` for invalid input.
- The function should not throw an error or raise an exception when given a valid configuration but no key file.
- The function's behavior should remain consistent across different test environments.
- No unexpected side effects should occur when loading the HMAC key without a file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/hashing.py	2 lines (ranges: 73-74)

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

AI ASSESSMENT

Scenario: Test the aggregation configuration defaults.

Why Needed: This test ensures that aggregation has sensible defaults, preventing 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 `capture_failed_output` default value for the integration gate is set to `False`.

Why Needed: This test prevents a potential regression where the default capture output is enabled by mistake.

Key Assertions:

- config.capture_failed_output should be `None` or `False`.
- The `capture_failed_output` configuration option does not have any effect on the integration gate's behavior.
- The integration gate will behave correctly when `capture_failed_output` is set to `True`.
- Setting `capture_failed_output` to `False` ensures that captured output is opt-in.
- If `capture_failed_output` is set to `True`, no test assertions are performed.
- The default value for `capture_failed_output` should be consistent across all integration gate configurations.
- Other configuration options, such as `capture_failed_output` itself, do not affect the integration gate'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/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the context mode defaults to 'none', which may cause issues with certain integration gate configurations.

Key Assertions:

- The function get_default_config() returns an instance of Config with llm_context_mode set to 'minimal'.
- The value of config.llm_context_mode is equal to 'minimal'.
- If the context mode defaults to 'none', then config.llm_context_mode should be set to 'minimal'.
- If the context mode is not 'minimal' when it's set, then config.llm_context_mode should not be 'minimal'.
- The function get_default_config() returns an instance of Config with llm_context_mode set to a value other than 'minimal'.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents a regression where LLM is enabled by default due to a misconfiguration or bug.

Key Assertions:

- config.is_llm_enabled() should return False
- config.get_llm_enabled_value() should be False
- get_default_config().llm_enabled() 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/options.py	4 lines (ranges: 107, 147, 224, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the `TestConfigDefaults` class's `omit_tests_from_coverage` attribute is set to `True` by default.

Why Needed: This test prevents a regression where the coverage threshold is not correctly applied when omitting tests from coverage.

Key Assertions:

- config.omit_tests_from_coverage is True
- assert config.omit_tests_from_coverage == True
- config.omit_tests_from_coverage should be set to `True` by default

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the provider defaults to 'none' when no value is provided.

Why Needed: Prevents a potential bug where the provider is set to an invalid or unexpected value.

Key Assertions:

- The `config.provider` attribute should be equal to 'none'.
- The `provider` attribute of the configuration object should have a value of 'none'.
- The `get_default_config()` function returns a configuration object with a `provider` attribute set to 'none'.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The function `get_default_config()` returns an object with an attribute `llm_context_exclude_globs` that contains a list of globs to exclude.
- Any string in the list of globs is either 'secret' or ends with '.env'.
- The function `any()` checks if any element in the list matches the specified condition (either 'secret' or ending with '.env').
- The function `any()` returns True as soon as it finds a match, rather than checking all elements in the list.
- Without this test, there's a possibility that sensitive information could be accidentally included in the LLM context due to a programming error.
- This test ensures that the default configuration is correct and does not include any secret files by default.
- The function `get_default_config()` is called with no arguments, which returns an object with the specified attributes.
- The attribute `llm_context_exclude_globs` is checked for being a list of strings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 10ms 5

AI ASSESSMENT

Scenario: Verifies that the test_deterministic_output function reports deterministic output.

Why Needed: This test prevents regression when the test_deterministic_output function is updated to use a different sorting algorithm.

Key Assertions:

- The nodeids in the report.json file are sorted by nodeid.
- The nodeids in the report.json file are not duplicated.
- The nodeids in the report.json file contain only unique values.
- The nodeids in the report.json file are sorted correctly even if there are duplicate nodeids.
- The nodeids in the report.json file are sorted alphabetically by nodeid.
- The nodeids in the report.json file do not contain any duplicates or duplicates with different sorting order.
- The nodeids in the report.json file contain only unique values and are sorted correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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` 10ms 5

AI ASSESSMENT

Scenario: Testing an empty test suite produces a valid report.

Why Needed: This test prevents regression by ensuring that an empty test suite generates a correct report.

Key Assertions:

- The total count of tests in the report is zero.
- There are no failed tests in the report.
- All test suites have been successfully written to the report file.
- The report contains only one section with 'total' key set to zero.
- No summary or other sections are present in the report.
- The report does not contain any duplicate keys.
- Each test is properly formatted and does not exceed 100 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/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` 34ms 5

AI ASSESSMENT

Scenario: The full pipeline generates an HTML report for all test cases.

Why Needed: This test prevents regression if the full pipeline is not generating a report for some tests, potentially due to a bug or configuration issue.

Key Assertions:

- The file "report.html" exists at the specified path.
- The content of "report.html" contains the string '
- The content of "report.html" contains the string 'test_pass', which is expected for passed test cases.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Test the full pipeline's ability to generate a valid JSON report.

Why Needed: This test prevents regression or bugs that may cause the generation of an invalid JSON report.

Key Assertions:

- The report should contain a 'schema_version' key with the correct value.
- The report should have a 'summary' section with the expected number of tests, passed, failed, and skipped.
- All test results should be included in the report.
- The report path should exist at the specified location.
- The JSON data should contain the required keys ('schema_version', 'summary', 'total', 'passed', 'failed', 'skipped')
- Test results should have the expected outcome (1 passed, 2 failed, and 0 skipped).

COVERAGE

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

	315, 317-322, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

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

AI ASSESSMENT

Scenario: The test verifies that the ReportRoot object has all required fields.

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

Key Assertions:

- schema_version
- run_meta
- summary
- 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	54 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where the 'is_aggregated' or 'run_count' fields are missing from the RunMeta object, potentially leading to incorrect analysis results.

Key Assertions:

- The 'is_aggregated' field is present in the data.
- The 'run_count' field is present in the data.
- The aggregation policy information ('run_count') is included when 'is_aggregated' 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/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 test run meta object contains all required status fields.

Why Needed: This test prevents a potential regression where the `RunMeta` object is not properly initialized with status fields, potentially causing issues downstream in the integration process.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents regressions where the schema version is not defined or does not match a semver-like format.

Key Assertions:

- The `SCHEMA_VERSION` attribute of the TestSchema object should be set to a valid version string.
- The `SCHEMA_VERSION` attribute should contain at least one dot (`.`) character, indicating that it is semver-like.
- The `SCHEMA_VERSION` attribute should match a valid semver-like format according to the SemVer specification.
- If the schema version does not contain a dot, it should be replaced with an empty string or a default value (e.g., `'0.1.0'`).
- If the schema version is less than 1.0.0, it should be updated to the next available semver release.
- If the schema version is greater than 2.0.0, it should be updated to the latest available semver release.
- The `SCHEMA_VERSION` attribute should not change between test runs or when the TestSchema object is cloned.
- If the `SCHEMA_VERSION` attribute is set to a value that is not compatible with the current version of the application (e.g., 1.0.2), it should be updated to the latest available semver release.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The `TestSchemaCompatibility` class is tested to ensure that it correctly validates the presence of required fields in a test case.

Why Needed: This test prevents regressions where a test case might not have all necessary fields for schema compatibility checks.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the LLM model does not return a provider instance for the 'gemini' configuration.

Key Assertions:

- The function `get_provider(config)` should return an instance of `GeminiProvider`.
- The class name of the returned provider instance should be `'GeminiProvider'`.
- The method `__class__.__name__` of the returned provider instance should match `'GeminiProvider'`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	10 lines (ranges: 52-53, 245, 247, 249, 252, 257, 262-263, 265)
src/pytest_llm_report/llm/gemini.py	7 lines (ranges: 134, 136-139, 141-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the provider is not correctly identified as 'LiteLLMProvider', potentially leading to incorrect model selection or usage.

Key Assertions:

- The `get_provider` function should return an instance of `LiteLLMProvider` when the 'provider' parameter is set to 'litellm'.
- The provider name should match 'LiteLLMProvider'.
- The class name of the returned object should be 'LiteLLMProvider'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: test_get_provider_with_none_config returns NoopProvider.

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

Key Assertions:

- The `get_provider` function should return an instance of `NoopProvider` when the `provider` parameter is set to 'none'.
- The `provider` attribute of the returned `NoopProvider` instance should be `None`.
- The `Config` class's `provider` parameter should not be used in this test.
- The `get_provider` function should raise an error when given a non-string provider value.
- The `get_provider` function should return `None` when the provided configuration is invalid.
- The `NoopProvider` instance should have no attributes or methods.
- The `provider` attribute of the `NoopProvider` instance should be `None`.
- The `Config` class's `provider` parameter should not be used in this test.

COVERAGE

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

Why Needed: This test prevents a potential bug where the correct provider class is not imported due to an import error.

Key Assertions:

- provider.__class__.__name__ == 'OllamaProvider'
- provider is of type OllamaProvider
- The provider instance has a valid __class__.__name__ 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	8 lines (ranges: 52-53, 245, 247, 249, 252-253, 255)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the `get_provider` function raises a `ValueError` when an unknown provider is provided.

Why Needed: To prevent unexpected behavior where an unknown provider might be used without proper validation.

Key Assertions:

- The `get_provider` function should raise a `ValueError` with the message 'unknown' when called with an unknown provider.
- The error message should include the string 'unknown'.
- The error message should not contain any other strings besides 'unknown'.
- The error message should be case-insensitive (e.g., 'Unknown', 'unknown').
- The `Config` object passed to `get_provider` should have a valid provider.
- The `Config` object passed to `get_provider` should have a provider that is not 'unknown'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that NoopProvider implements LlmProvider interface.

Why Needed: Prevents regression in case of NoopProvider not implementing required methods.

Key Assertions:

- The provider should have the required methods (annotate, is_available, get_model_name, config).

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the provider does not return any annotation even if it's supposed to be empty.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the NoopProvider returns an empty string when given an empty configuration.

Why Needed: This test prevents a regression where the model name is not returned correctly when the config is empty.

Key Assertions:

- The function provider.get_model_name() should return an empty string.
- The function provider.get_model_name() should be called with an argument that is an empty Config object.
- The function provider.get_model_name() should raise a ValueError with an appropriate message when given an 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/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 66)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the NoopProvider instance is always available.

Why Needed: This test prevents a potential regression where the provider might not be available due to configuration issues.

Key Assertions:

- The `is_available()` method should return True for all instances of the NoopProvider.
- The `is_available()` method should raise an AssertionError if it returns False.
- The `is_available()` method should not throw any exceptions when called on a valid instance of the NoopProvider.
- The `is_available()` method should behave consistently across different configurations.
- The `is_available()` method should return True for all instances with default configuration.
- The `is_available()` method should return False for instances with custom configuration that disables availability.
- The `is_available()` method should not be affected by the presence of any exceptions in its implementation.

COVERAGE

src/pytest_llm_report/collector.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: The test verifies that the annotation summary is printed when annotations run.

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

Key Assertions:

- The function `get_provider` from `pytest_llm_report.llm.annotator` returns a `FakeProvider` instance.
- The `annotate_tests` function calls `get_provider` with the correct configuration.
- The captured output contains the expected string 'Annotated 1 test(s) via litellm'.
- The `test_annotate_tests_emits_summary` function is called with the correct arguments and returns a `TestCaseResult` instance.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the LLM annotator reports progress via a callback.

Why Needed: This test prevents regression where the LLM annotator does not report progress, potentially causing issues with test reporting.

Key Assertions:

- The test verifies that the LLM annotation starts and ends with the correct messages.
- The test checks if the correct message is appended to the list of messages.
- The test verifies that the correct scenario is included in the message.
- The test checks if the progress callback is set correctly.
- The test verifies that the progress callback appends the correct message to the list.
- The test checks if the correct node ID is included in the message.
- The test verifies that the LLM annotation progresses as expected.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

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

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

Key Assertions:

- The 'tests/test_a.py::test_a' node should be included in the LLM annotation.
- The 'tests/test_b.py::test_b' node should be excluded from the LLM annotation due to the 'llm_opt_out=True' parameter.
- The 'tests/test_c.py::test_c' node should not be included in the LLM annotation.
- The number of nodes in the LLM annotation should not exceed 1 (the maximum limit set by llm_max_tests=1).
- The 'tests/test_a.py::test_a' node should have an LLM annotation, while the other two nodes should not.
- The 'tests/test_b.py::test_b' and 'tests/test_c.py::test_c' nodes should not have any LLM annotations.
- The number of LLM annotations should be 1 (the maximum allowed).
- The 'tests/test_a.py::test_a' node should be the only one included in the LLM annotation.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	65 lines (ranges: 45, 48-49, 56-57, 59, 61-62, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the LLM annotator exceeds the allowed requests per minute.

Key Assertions:

- The provider's `calls` attribute should contain the expected list of node IDs.
- The `sleep_calls` attribute should contain the expected sleep duration in seconds.
- The `provider.calls` attribute should match the expected list of node IDs.
- The `sleep_calls` attribute should not exceed 2.0 seconds.
- The `provider.calls` attribute should be an array containing only the expected node ID.
- The `time.sleep(2.0)` call should have been made at least once during the test execution.
- The `time.sleep(0.0)` call should have been made before the first `time.sleep(2.0)` call.
- The `time.sleep` function should not be called more than 30 times in a minute.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-173, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

Scenario: Test that annotation fails when unavailable provider is specified.

Why Needed: To prevent skipping of annotation tests due to unavailability of providers.

Key Assertions:

- The function `llm.annotator.get_provider` should be called with a valid configuration.
- The `is_available` method of the `UnavailableProvider` class should return False.
- The message 'is not available' should be captured in the output of `capsys.readouterr()`.

COVERAGE

src/pytest_llm_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: Test that annotations are cached between runs and that the annotation is correctly retrieved.

Why Needed: This test prevents regression where the annotation is lost after a cache flush or restart.

Key Assertions:

- The `provider` attribute of the `TestCaseResult` instance should be set to `tests/test_sample.py::test_case` before calling `get_provider`.
- The `llm_annotation` attribute of the `TestCaseResult` instance should not be `None` after calling `annotate_tests` with a cached provider.
- The scenario of the annotation in the `TestCaseResult` instance should match the scenario specified by the cache directory.
- The `provider` attribute of the `TestCaseResult` instance should be set to `tests/test_sample.py::test_case` before calling `get_provider` again after setting it to a different provider.
- The `llm_annotation` attribute of the `TestCaseResult` instance should not be `None` after calling `annotate_tests` with a non-cached provider.

COVERAGE

src/pytest_llm_report/cache.py	30 lines (ranges: 39-41, 53, 55-56, 58, 60-62, 68-73, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-81, 87-92, 97-98, 100, 102, 104, 115-122, 127, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the schema is not enforced correctly, potentially leading to incorrect validation of contracts.

Key Assertions:

- required
- scenario
- 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/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the `AnnotationSchema` can correctly parse a dictionary containing required fields.

Why Needed: This test prevents potential bugs in the `AnnotationSchema` where it may not be able to accurately determine the schema from a dictionary, potentially leading to incorrect or missing key assertions.

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: An annotation schema is created with an empty dictionary.

Why Needed: The test prevents a potential bug where the schema creation process fails when given an empty input.

Key Assertions:

- schema.scenario = "" (empty string)
- schema.why_needed = "" (empty string) (to prevent schema creation failure)
- schema.scenario == "" (checks that the scenario is set to an empty string)
- schema.why_needed == "" (checks that the why needed message is set to an empty string)
- schema.scenario != "" (checks that the scenario is not set to a non-empty string)
- schema.why_needed != "" (checks that the why needed message is not set to a non-empty string)
- schema.scenario == "" (checks that the schema has an empty string as its value)

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the AnnotationSchema class correctly handles partial inputs by verifying if it matches the expected scenario.

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

Key Assertions:

- schema.scenario == 'Partial only'
- assert schema.why_needed == ''
- schema.scenario should be equal to 'Partial only' according to the expected annotation
- schema.why_needed should be an empty string according to the expected annotation
- The AnnotationSchema class correctly handles partial inputs by verifying if it matches the expected scenario
- The AssertionSchema class correctly handles partial inputs by verifying if it matches the expected scenario

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the annotation schema is missing required fields, potentially causing errors or inconsistencies during validation.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents regression by ensuring that the AnnotationSchema instance can be serialized and deserialized correctly.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 90-92, 94-96, 98)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the Factory returns a NoopProvider when the provider is set to 'none'.

Why Needed: This test prevents a regression where the Factory returns an incorrect provider for the 'none' provider.

Key Assertions:

- The function `get_provider(config)` should return a NoopProvider instance.
- The `provider` attribute of the returned `NoopProvider` instance should be set to 'none'.
- The `provider` attribute of the returned `NoopProvider` instance should not be set to any other provider.
- The `provider` attribute of the returned `NoopProvider` instance should have a value of 'none'.
- The `provider` attribute of the returned `NoopProvider` instance should not have a value of an empty string or None.
- The `provider` attribute of the returned `NoopProvider` instance should not be set to any other provider.
- The `provider` attribute of the returned `NoopProvider` instance should have a value that is different from 'none'.
- If the Factory returns an incorrect provider, it should raise an assertion error.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: Verify that the `NoopProvider` class correctly inherits from `LlmProvider`.

Why Needed: Prevents a potential bug where the `NoopProvider` class is incorrectly implemented as an instance of `LlmProvider` instead of its intended interface.

Key Assertions:

- The `provider` variable should be an instance of `LlmProvider`.
- The `provider` variable should not have any additional attributes or methods beyond what is defined in the `LlmProvider` interface.
- The `provider` variable's type should match the expected type (`LlmProvider`) without any conversions or casts.
- No other attributes or methods of `provider` should be present, as they are not part of the `LlmProvider` interface.
- Any additional imports or dependencies required by the `NoopProvider` class should not affect its functionality in this test.
- The `config` object passed to `NoopProvider` should not have any unexpected side effects on the LLM provider implementation.
- The `noop` function provided with `NoopProvider` should be called correctly and without any errors.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the NoopProvider would incorrectly return an annotation for a test function with no annotations.

Key Assertions:

- assert result.scenario == "" (empty string)
- assert result.why_needed == "" (empty string)
- assert result.key_assertions == [] (no key assertions performed)

COVERAGE

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

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

AI ASSESSMENT

Scenario: Annotating a TestCaseResult object with the correct configuration and provider.

Why Needed: The test prevents regression in case of incorrect annotation configuration or provider.

Key Assertions:

- The `result` object has attributes `scenario`, `why_needed`, and `key_assertions`.
- The `result` object has the expected attributes for a TestCaseResult.
- The `result` object's `why_needed` attribute is set correctly based on the annotation configuration.
- The `result` object's `key_assertions` list contains the expected checks.
- The `result` object's `scenario` attribute is not empty.
- The `result` object's `why_needed` attribute does not contain any invalid values.
- The `result` object's `key_assertions` list does not contain any missing or invalid assertions.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The provider should return a valid result for the given input.
- The result should be non-null.
- The annotation outcome should be 'passed'.
- No exception should be raised when providing an empty code.
- The contract should not throw any errors or exceptions when handling empty code.
- The test should pass without any failures or errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 return a non-None value when given an annotation with `None` context.

Why Needed: This test prevents a potential bug where the `provider.annotate` method returns `None` due to the presence of `None` in the annotation context.

Key Assertions:

- The `result` variable should not be `None` after calling `provider.annotate(test, 'code', None)`.
- The `result` variable is expected to have a non-`None` value.
- The `provider.annotate` method should return a valid object (e.g., `TestCaseResult`) instead of `None`.
- The annotation context (`'code'`) should be properly propagated to the `TestCaseResult` instance.
- The test result should not be affected by the presence of `None` in the annotation context.
- The `test_result` object returned by `provider.annotate` should have a valid state (e.g., `passed`, `failed`, etc.)
- The test case should pass without any errors or exceptions when given an annotation with `None` context.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevent regression in the contract where providers are not annotated with a specific method.

Key Assertions:

- The provider has an attribute named 'annotate'.
- The provider is callable.
- The provider has an 'annotate' method.
- The provider's 'annotate' method is callable.
- All providers have the same 'annotate' method.
- The 'annotate' method is present in all tested providers.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 issue where the `annotate` method throws an exception when dealing with very large contexts.

Key Assertions:

- The `annotate` method should not throw an exception when given a context of size up to 1000.
- The `annotate` method should return an empty list for a context larger than 1000.
- The `annotate` method should raise a `ValueError` with a meaningful error message for a context larger than 1000.
- The `annotate` method should not throw an exception when given a context of size 0.
- The `annotate` method should return the expected result (an empty list) for a context of size 1.
- The `annotate` method should raise a `ValueError` with a meaningful error message for a context larger than 1000.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	9 lines (ranges: 52-53, 72, 75-76, 80, 165, 167, 175)
src/pytest_llm_report/llm/gemini.py	155 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-221, 233, 245-248, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 346, 348-350, 352-355, 360-363, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417-418, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that LiteLLMProvider annotates missing dependencies correctly.

Why Needed: This test prevents a potential bug where the provider does not report missing dependencies and instead installs them silently.

Key Assertions:

- The annotation message is correct and informative.
- The error message includes the name of the missing dependency (litellm).
- The annotation message provides instructions on how to install the missing dependency (pip install litellm).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 `annotate` method fails when an API token is missing.

Why Needed: This test prevents a potential bug where the `annotate` method throws an error due to an unprovided API token.

Key Assertions:

- The `annotation.error` attribute should be set to 'GEMINI_API_TOKEN is not set'.
- The `provider.annotate(test, ...)` call should raise an exception.
- The `test_case()` function should throw a `AssertionError` with the message 'GEMINI_API_TOKEN is not set'.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The 'json' key in the captured dictionary contains the expected response data.
- The 'totalTokenCount' key in the captured dictionary matches the expected value.
- The 'candidates' list in the captured dictionary includes at least one record with a valid token count.
- The 'usageMetadata' dictionary contains the expected rate limits.
- The 'models?' query parameter is present in the URL and has the correct model name.
- The 'rateLimits?' query parameter is present in the URL and has the correct rate limit value.
- The 'tokensPerMinute?' query parameter is present in the URL and matches the expected value.
- The 'tokenUsage' list in the captured dictionary contains at least one record with a valid token count.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)

src/pytest_llm_report/llm/gemini.py	183 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-101, 103, 105, 107-109, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-223, 225-227, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-343, 346, 348-350, 352-355, 360-366, 368, 370-371, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `annotate` method fails to retry after rate limiting, causing the model to be stuck in an infinite loop.

Key Assertions:

- The `annotate` method should call `self._retry_on_rate_limit` before making the annotation request.
- The `annotate` method should call `self._retry_on_rate_limit` again after a certain amount of time (e.g. 1 second) if the initial retry fails.
- The `annotate` method should raise an exception when rate limiting occurs and retries are not allowed.
- The `annotate` method should log a warning message indicating that rate limiting occurred, but no retry was made.
- The `_retry_on_rate_limit` method should be called with the correct arguments (e.g. `self._rate_limit`, `self._max_retries`) when rate limiting occurs.
- The `_retry_on_rate_limit` method should call `self._logger.warning` to log a warning message indicating that rate limiting occurred, but no retry was made.
- The `annotate` method should not make any requests during the retry period (e.g. 1 second).
- The `annotate` method should raise an exception when it is called multiple times in quick succession while rate limiting occurs.

COVERAGE

src/pytest_llm_report/collector.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 used with a large number of annotations.

Why Needed: This test prevents a potential bug where the model rotation is not applied correctly to large datasets, leading to performance issues or incorrect results.

Key Assertions:

- The `annotate` method should rotate models on the daily limit for all annotations in the dataset.
- The number of rotated models should be equal to the total number of annotations minus one (for the last annotation),
- The model rotation should occur immediately after the first annotation, not after a certain delay or threshold.
- The `annotate` method should handle cases where there are multiple annotations with the same label correctly.
- The model rotation should be applied to all models in the dataset, regardless of their labels or types.
- The test should pass for both positive and negative annotations (e.g., 1000 positive annotations and -500 negative annotations).
- The `annotate` method should not rotate models on the daily limit if there are no annotations in the dataset.

COVERAGE

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

	391-393, 396-399, 401-402, 405, 408-410, 412-414, 417, 419-420, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the annotate function skips daily limits when provided with a valid provider.

Why Needed: This test prevents a bug where the annotate function fails to skip daily limits due to incorrect handling of the provider.

Key Assertions:

- The annotate function should not attempt to annotate data for providers with daily limits.
- The annotate function should return an error message indicating that daily limits are exceeded.
- The annotate function should update the provider's annotation status to 'skipped' when a valid provider is provided.
- The annotate function should not attempt to annotate data for providers without daily limits.
- The annotate function should return an error message indicating that no valid provider was found.
- The annotate function should update the provider's annotation status to 'skipped' when a valid provider with a daily limit is provided.
- The annotate function should only skip annotations for providers with daily limits, not for those without.
- The annotate function should handle cases where the provider has no daily limit or an invalid daily limit value.

COVERAGE

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

	346, 348-350, 352-355, 360-366, 368, 370, 372-377, 381-382, 385-387, 391-393, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that LiteLLM provider annotates a successful response with the correct key assertions and confidence level.

Why Needed: Prevents regression by ensuring that the annotation is correctly configured for success scenarios.

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 within 24 hours.

Key Assertions:

- The model's performance metrics (e.g. accuracy, F1 score) should return to normal within 24 hours of being exhausted.
- The model's training time should decrease by at least 50% after 24 hours of exhaustion.
- The model's inference time should increase by less than 20% after 24 hours of exhaustion.
- The model's memory usage should decrease by at least 30% after 24 hours of exhaustion.
- The model's GPU utilization should decrease by at least 40% after 24 hours of exhaustion.
- The model's disk I/O time should increase by less than 15% after 24 hours of exhaustion.
- The model's memory allocation should not exceed the available memory for more than 24 hours.
- The model's CPU utilization should not exceed 80% for more than 24 hours.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/base.py`	`26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)`

src/pytest_llm_report/llm/gemini.py	190 lines (ranges: 39-42, 45-46, 48-50, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-188, 190-191, 193-194, 196, 200-208, 210-211, 213-214, 217-222, 225-227, 252-254, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-341, 343, 346, 348-350, 352-355, 360-366, 368, 370, 372-377, 381-382, 385-387, 391-393, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the `fetch_available_models` method throws an error when no models are available.

Why Needed: This test prevents a potential regression where the `fetch_available_models` method might return incorrect results or raise an exception due to a missing model.

Key Assertions:

- The `fetch_available_models` method raises a `ValueError` with a meaningful error message when no models are available.
- The method does not throw an exception when all models are available.
- The method returns the correct list of available models when all models are available.
- The method throws an exception with a specific error message when all models are available.
- The `fetch_available_models` method raises an exception with a more informative error message than `ValueError`.
- The `fetch_available_models` method does not raise an exception when the input is invalid or empty.
- The `fetch_available_models` method returns an empty list when all models 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	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 10 seconds.

Why Needed: This test prevents a potential issue where the model list does not update after a certain time period, potentially causing unexpected behavior or errors.

Key Assertions:

- model_list is updated with new models within the specified interval
- no stale models are present in the model list
- the model list size remains constant despite changes to the underlying data 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/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 annotation does not surface completion errors.

Key Assertions:

- The annotation should contain an error message indicating a completion error.
- The annotation should raise a RuntimeError when encountering a completion error.
- The annotation should include the string 'boom' in its error message.
- The annotation should not ignore completion errors and instead surface them correctly.
- The annotation should not silently fail when encountering a completion error.

COVERAGE

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

Why Needed: To prevent unexpected behavior when invalid key_assertions are provided to the annotate method.

Key Assertions:

- The 'key_assertions' parameter must be a list of strings.
- Invalid response: key_assertions must be a list
- Key assertions should not contain any string values (e.g., None, empty strings).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	22 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207-209)
src/pytest_llm_report/llm/litellm_provider.py	25 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 66-67, 69, 73, 76, 81-82, 84-85)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The LiteLLMProvider should report a missing dependency when trying to annotate a case where the provider is not installed.

Why Needed: This test prevents a potential bug where the LiteLLMProvider does not correctly handle cases where it is not installed, leading to incorrect error messages.

Key Assertions:

- The annotation will contain an error message indicating that the `litellm` dependency is missing and how to install it.
- The annotation will include a specific URL pointing to the installation instructions for `litellm`.
- The annotation will correctly identify the provider as 'litellm' even if it has not been installed.
- The annotation will not report any other error messages or warnings when trying to annotate a case where the provider is missing.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/litellm_provider.py	5 lines (ranges: 37-41)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that LiteLLMProvider annotates a successful response with the correct key assertions and confidence level.

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

Key Assertions:

- status ok
- redirect
- confidence >= 0.8

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the provider does not detect installed modules, potentially leading to unexpected behavior or errors.

Key Assertions:

- The `is_available()` method of the `LiteLLMProvider` instance should return `True` when the `litellm` module is available in the system's module list.
- The `sys.modules` dictionary should contain a reference to the `fake_litellm` SimpleNamespace created during setup.
- The `is_available()` method should not raise an exception or throw any other error when the `litellm` module is installed.
- The provider's configuration should be able to detect the installed `litellm` module without requiring manual intervention.
- The test should pass even if the `litellm` module is installed but not imported directly (e.g., as a package).
- The provider's behavior should remain consistent across different Python versions 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/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 fallbacks on context length error are correctly handled.

Why Needed: This test prevents a potential regression where the LLM provider may not be able to handle context length errors properly.

Key Assertions:

- When an invalid context is provided, it should throw a ContextLengthError with a meaningful message.
- The annotate method should raise a ValueError when the context length exceeds the maximum allowed value.
- The error message should include information about the original input that caused the error.
- The test should be able to reproduce the error using different inputs (e.g., different contexts, different LLM providers).
- The error should not be silently ignored or masked by other parts of the code.
- The annotate method should handle context length errors in a way that is consistent with the expected behavior for valid inputs.
- The test should pass even when the LLM provider returns an error response (e.g., 500 Internal Server Error).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: Verifies that the annotate method returns an error message when a call to Ollama raises a RuntimeError.

Why Needed: This test prevents regression in handling call errors by ensuring that the annotation function correctly identifies and reports any errors that occur during the annotation process.

Key Assertions:

- The annotation function should return the string 'Failed after 3 retries. Last error: boom' as its error message when a call to Ollama raises a RuntimeError.
- The annotation function should not report an error if the call to Ollama is successful.
- The annotation function should correctly handle multiple retries and update the error message accordingly.

COVERAGE

src/pytest_llm_report/collector.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: Test that OllamaProvider annotates missing httpx dependency correctly.

Why Needed: This test prevents a regression where the provider incorrectly reports an error message for missing httpx, leading to incorrect user experience.

Key Assertions:

- The annotation error is set to 'httpx not installed. Install with: pip install httpx'.
- The provider correctly reports that the httpx dependency is missing.
- The test passes if the annotation error matches this expected message.
- The test fails if the annotation error does not match this expected message.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test Ollama provider full annotation flow with mocked HTTP to test annotate success for full flow

Why Needed: Prevents regression in annotating full LLM flow due to potential issues with status checks and token validation

Key Assertions:

- check status of response after successful login
- validate token sent by Ollama provider during annotation process
- assert that the annotated function returns True without any errors

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The 'url' captured in the captured dictionary is set to 'http://localhost:11434/api/generate'.
- The 'json' captured in the captured dictionary contains the correct model and prompt data.
- The 'system' captured in the captured dictionary contains the correct system prompt data.
- The 'stream' captured in the captured dictionary is False.
- The API call timeout is set to 60 seconds as configured in the provider settings.
- The response from OLLAMA matches the expected response.

COVERAGE

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

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

Key Assertions:

- The captured response from the API should contain an empty string as the 'model' key.
- The captured response from the API should contain 'llama3.2' as the 'model' value.
- The captured response from the API should not contain any other values for the 'model' key.
- The captured response from the API should be an object with a single property 'model' of type str.
- The captured response from the API should have no other properties or values in the 'model' field.
- The captured response from the API should not contain any additional keys or values for the 'model' key.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 bug where the provider incorrectly assumes the server is running and always returns True for availability checks.

Key Assertions:

- The `_check_availability()` method of the `OllamaProvider` instance should return `False` when the server is not available.
- The `Config` instance passed to the `OllamaProvider` constructor should have a `server_url` attribute that points to a non-existent URL.
- The `get()` method called on the `fake_httpx` object raised as a `ConnectionError` when it attempts to connect to the server.
- The `httpx` module imported from the `sys.modules` dictionary is not set to point to the fake HTTPX instance.
- The `_check_availability()` method of the `OllamaProvider` instance should raise an exception or return a specific value indicating failure when the server is unavailable.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	6 lines (ranges: 87-88, 90-91, 93-94)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- The method _check_availability() of the OllamaProvider instance is called with no arguments.
- The status code returned by the FakeResponse object is not equal to 200.
- The provider's availability is set to False using the monkeypatched sys.modules['httpx'].
- The provider's availability is checked using the _check_availability() method.
- The assertion that provider._check_availability() returns a boolean value (True or False) is performed.
- The assertion that provider._check_availability() returns 200 is not performed because it does not match the expected status code.
- The assertion that provider._check_availability() returns False is performed.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the Ollama provider checks availability via /api/tags endpoint successfully.

Why Needed: Prevents a potential bug where the provider does not check for availability before returning data.

Key Assertions:

- The provider should return True when checking for availability.
- The provider should return False when checking for inactivity.
- The provider's response status code should be 200 (OK) when checking for tags.
- The provider's response body should contain the expected list of available Ollama models.
- The provider's response headers should contain the expected 'Content-Type' and 'X-Content-Type-Options: nosniff' headers.
- The provider's response status code should be 200 (OK) when checking for tags with a specific tag name.
- The provider's response body should contain the expected list of available Ollama models with the specified tag name.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/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 bug where the provider incorrectly returns `False` for local configurations.

Key Assertions:

- provider.is_local() == True
- provider.config.provider == 'ollama'
- provider.config is not None
- not provider.config.provider == 'unknown' or 'non-ollama'
- config is a valid instance of Config with provider set to 'ollama'
- is_local() should be True for local configurations
- is_local() should return False for non-local configurations

COVERAGE

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

- The expected error message is 'Failed to parse LLM response as JSON'.
- The `annotation.error` attribute contains the correct error message.
- The `provider._parse_response()` method returns an instance of `OllamaProviderError` with the specified error message.
- The `OllamaProvider` class has a docstring that explains why it throws this error.
- The test verifies that the provider correctly reports invalid JSON responses.

COVERAGE

`src/pytest_llm_report/collector.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 Ollama provider rejects invalid key_assertions payloads.

Why Needed: To prevent the Ollama provider from incorrectly handling or rejecting malformed key_assertions payloads.

Key Assertions:

- must be a list
- contains only valid keys

COVERAGE

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

Why Needed: This test prevents a potential bug where the provider incorrectly extracts JSON or fails to parse certain types of data.

Key Assertions:

- response_json_is_valid
- provider_returns_code_fence
- provider_extracts_correct_data
- provider_does_not_return_empty_string
- provider_returns_non_empty_string_if_missing
- provider_returns_properly_formatted_json
- provider_extracts_correct_key_value_pairs
- provider_extracts_correct_object_types

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	6 lines (ranges: 38, 42-44, 46-47)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The provided test verifies that the Ollama provider correctly parses a JSON response in a plain markdown fence without any language specification.

Why Needed: This test prevents potential issues where the provider may incorrectly parse or ignore non-language marked-up text as JSON.

Key Assertions:

- The `response` variable is assigned a string containing a valid JSON representation of a plain markdown fence.
- The `provider.parse_response_json(response)` call returns True, indicating that the JSON was successfully parsed.
- The `response` variable contains only whitespace characters and Markdown syntax (e.g., ```, ``), which should be ignored by the provider.
- The `config.provider` attribute is set to 'ollama', which indicates that the provider has been configured with an Ollama instance.
- The `provider.config` attribute provides access to the configuration object, allowing for customization of the provider's behavior.
- The `provider.parse_response_json_in_plain_fence` method is called with a valid JSON response, demonstrating its effectiveness in this scenario.
- Any errors or exceptions raised during the parsing process are likely to be caught and handled by the provider's error handling mechanisms.
- The test ensures that the provider can handle various edge cases, such as malformed or incomplete JSON responses.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	6 lines (ranges: 38, 42-44, 46-47)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test Ollama provider parses valid JSON responses with success.

Why Needed: Prevents bugs that occur when parsing invalid or malformed JSON responses.

Key Assertions:

- assert a is not None
- assert b is not None
- assert response_data['scenario'] == 'Tests feature'
- assert response_data['why_needed'] == 'Stops bugs'
- assert annotation.confidence == 0.8

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the serialized representation of `CoverageEntry` is incorrect, potentially leading to data corruption or inconsistencies in the test results.

Key Assertions:

- The 'file_path' key should contain the expected value.
- The 'line_ranges' key should contain the expected string representation.
- The 'line_count' key should contain the expected integer value.

COVERAGE

src/pytest_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()` returns the expected data structure.

Why Needed: This test prevents a potential bug where the `CoverageEntry` object is not properly serialized to JSON.

Key Assertions:

- The 'file_path' key should contain the correct value.
- The 'line_ranges' key should contain the correct values.
- The 'line_count' key should contain the correct value.
- The 'coverage' key should be missing.
- The 'start' and 'end' keys for line ranges should be present with correct values.
- All required keys ('file_path', 'line_ranges', 'line_count') should be included in the output JSON.

COVERAGE

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

Why Needed: This test prevents regression when updating the `CoverageEntry` class to include additional fields in its serialization.

Key Assertions:

- The 'file_path' field of the serialized CoverageEntry should match the original file path.
- The 'line_ranges' field of the serialized CoverageEntry should match the expected line ranges.
- The 'line_count' field of the serialized CoverageEntry should match the original value.
- Additional fields (if any) in the `CoverageEntry` class should not be included in the serialization.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where an empty annotation would have no confidence or error value.

Key Assertions:

- The `annotation` object has the expected attributes: `scenario`, `why_needed`, and `error`.
- The `confidence` attribute is set to `None` as it should be for an empty annotation.
- The `error` attribute is also set to `None` as it should be for an empty annotation.
- The `key_assertions` list is empty, indicating that the test does not verify any specific key assertions.
- The `annotation` object has no attributes other than `scenario`, `why_needed`, and `error`.
- The `confidence` attribute is missing from the `annotation` object.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the `to_dict` method of `LlmAnnotation` returns a dictionary with required fields.

Why Needed: This test prevents regression by ensuring that the minimal annotation format includes all necessary information.

Key Assertions:

- The dictionary must contain 'scenario' key.
- The dictionary must contain 'why_needed' key.
- The dictionary must contain 'key_assertions' key.
- The dictionary should not contain 'confidence' key when it is None.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test to dictionary with all fields

Why Needed: Prevents LlmAnnotation model from being misconfigured or incorrectly annotated.

Key Assertions:

- The 'scenario' field is correctly set to the expected value.
- The 'confidence' field matches the expected confidence level (0.95).
- The 'context_summary' dictionary contains the correct mode and bytes 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 has expected schema version and empty lists.

Why Needed: Prevents a potential bug where the report is missing the required 'schema_version' key or contains non-empty 'tests', 'warnings', or 'collection_errors' lists.

Key Assertions:

- assert d['schema_version'] == SCHEMA_VERSION
- assert d['tests'] == []
- assert 'warnings' not in d
- assert 'collection_errors' not in d

COVERAGE

src/pytest_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 to include collection errors.

Why Needed: This test prevents a regression where the report does not accurately reflect collection errors.

Key Assertions:

- The 'collection_errors' key in the report dictionary should contain exactly one item.
- The value of the 'nodeid' key in the first item of the 'collection_errors' list should be 'test_bad.py'.
- All items in the 'collection_errors' list should have a non-empty 'nodeid' attribute.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test reports with warnings should be handled correctly.

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

Key Assertions:

- The length of `d['warnings']` is equal to 1.
- The code in `d['warnings'][0]` matches 'W001'.
- All warnings 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	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 regressions where the order of tests is not guaranteed to match their original nodeids.

Key Assertions:

- The list of nodeids matches the expected order.
- All nodeids are present in the list.
- Nodeids are sorted correctly (a → m, a → z).
- No duplicate nodeids are present in the list.
- All tests have unique nodeids.
- Test nodes are not duplicated in the list.
- The order of tests is preserved.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the `to_dict` method of `ReportWarning` returns a dictionary with 'detail' key.

Why Needed: This test prevents a potential issue where the detailed warning message is not included in the dictionary returned by the `to_dict` method, potentially causing confusion or errors when working with warnings.

Key Assertions:

- The value of 'detail' in the dictionary returned by `warning.to_dict()` should be '/path/to/file'.
- The key 'detail' exists in the dictionary returned by `warning.to_dict()` and its value is indeed '/path/to/file'.
- The value of 'detail' in the dictionary returned by `warning.to_dict()` does not contain any additional information.
- The warning object passed to `ReportWarning` has a valid `to_dict` method that returns a dictionary with the required keys.
- The `to_dict` method of `ReportWarning` is correctly implemented and does not return an empty dictionary or raise an exception.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test 'test_to_dict_without_detail' verifies that a ReportWarning instance can be converted to a dictionary without including its detail information.

Why Needed: This test prevents a warning about missing detail from being included in the report.

Key Assertions:

- The 'code' key should contain the expected value 'W001'.
- The 'message' key should contain the expected value 'No coverage'.
- The 'detail' key should be absent from the dictionary.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that RunMeta has aggregation fields.

Why Needed: Prevents regression where RunMeta is missing required aggregation fields.

Key Assertions:

- assert d['run_id'] == 'run-123'
- assert d['run_group_id'] == 'group-456'
- assert d['is_aggregated'] is True
- assert d['aggregation_policy'] == 'merge'
- assert d['run_count'] == 3
- assert len(d['source_reports']) == 2

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	39 lines (ranges: 277-279, 281-283, 364-380, 382, 385, 387, 390, 393, 395, 397, 399-405, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that LLM fields are excluded when annotations are disabled.

Why Needed: Prevents regression where LLMs' annotation settings are not properly handled when annotations are disabled.

Key Assertions:

- The 'llm_annotations_enabled' key is present in the data.
- The 'llm_provider' key is present in the data.
- The 'llm_model' key is present in the data.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verifies that LLM traceability fields are included when enabled for a RunMeta object.

Why Needed: This test prevents regression in the case where LLMs are not properly configured or annotated, potentially leading to incorrect results or errors.

Key Assertions:

- The value of `llm_annotations_enabled` is True.
- The value of `llm_provider` is set to 'ollama'.
- The value of `llm_model` is set to 'llama3.2:1b'.
- The value of `llm_context_mode` is set to 'complete'.
- The value of `llm_annotations_count` is 10.
- The value of `llm_annotations_errors` is 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: Verifies that non-aggregated reports do not include source_reports.

Why Needed: Prevents regression where non-aggregated reports incorrectly include source_reports.

Key Assertions:

- The 'source_reports' key should be absent from the report dictionary.
- The 'is_aggregated' value should be set to False for non-aggregated reports.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test RunMeta to dict with all optional fields.

Why Needed: Prevents regression in RunMeta serialization when including legacy fields.

Key Assertions:

- Verify that the 'git_sha' field is correctly set to 'abc1234'.
- Assert that 'git_dirty' is True.
- Verify that 'repo_version' and 'repo_git_sha' are correctly set to '1.0.0'.
- Assert that 'repo_git_dirty' is False.
- Verify that 'plugin_git_sha' is correctly set to 'def5678'.
- Assert that 'plugin_git_dirty' is False.
- Verify the length of 'source_reports' is 1 as expected.
- Verify all required fields are present in the output 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	49 lines (ranges: 277-279, 281-283, 364-380, 382-405, 407, 419)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test RunMeta to include run status fields.

Why Needed: The test prevents a regression where the 'RunMeta' object does not contain all necessary run status fields.

Key Assertions:

- The 'exit_code' field is set to 1.
- The 'interrupted' field is True.
- The 'collect_only' field is True.
- The 'collected_count' field equals 10.
- The 'selected_count' field equals 8.
- The 'deselected_count' field equals 2.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The schema version should be split into three parts (e.g., '1.2.3')
- Each part of the schema version should be a digit (0-9)
- The first part of the schema version should not be zero (e.g., '1', not '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_models.py::TestSchemaVersion::test_schema_version_in_report_root` 1ms 3

AI ASSESSMENT

Scenario: Test that the `ReportRoot` class includes the schema version in its report root.

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

Key Assertions:

- The `schema_version` attribute of the `ReportRoot` instance should be equal to `SCHEMA_VERSION`.
- The value of `schema_version` should match the actual schema version stored in the report root.
- The `to_dict()` method of the `ReportRoot` instance should return a dictionary with a `schema_version` key that matches `SCHEMA_VERSION`.

COVERAGE

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

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

AI ASSESSMENT

Scenario: CoverageEntry should serialize correctly.

Why Needed: This test prevents a bug where the coverage entry's line ranges are not properly serialized to JSON.

Key Assertions:

- The 'file_path' key is set to 'src/foo.py'.
- The 'line_ranges' key is set to '1-3, 5, 10-15'.
- The 'line_count' key 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`	`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 bugs or regressions where the minimal annotation is missing certain required fields.

Key Assertions:

- The 'scenario' field should be present in the dictionary.
- The 'why_needed' field should be present in the dictionary.
- The 'key_assertions' field 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	5 lines (ranges: 277-279, 281, 283)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test SourceReport to_dict_with_run_id verifies that the 'run_id' key is present in the output dictionary.

Why Needed: This test prevents a potential bug where the 'run_id' field is not included in the output dictionary, potentially causing incorrect data analysis.

Key Assertions:

- The 'run_id' key should be present in the output dictionary.
- The value of the 'run_id' key should match the provided run_id string ('run-1').
- The 'run_id' field should not be missing from the output dictionary.
- The 'run_id' field should have the correct format (e.g., 'run-XX')
- The 'run_id' value should be a string and not an integer or other data type
- The 'run_id' value should match the provided run_id string ('run-1')

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents bugs or regressions where the serialization of `CoverageEntry` objects is incorrect, potentially leading to unexpected behavior or errors in downstream code.

Key Assertions:

- The 'file_path' key in the dictionary should match the expected value.
- The 'line_ranges' key in the dictionary should match the expected value.
- The 'line_count' key in the dictionary should match the expected value.
- The `to_dict()` method of `CoverageEntry` object returns a dictionary with the correct keys and values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: To ensure that the minimal result is correctly constructed and contains all necessary information.

Key Assertions:

- The 'nodeid' field should match the expected value.
- The 'outcome' field should be set to 'passed'.
- The 'duration' field should be set to 0.0 (or a default value).
- The 'phase' field should be set to 'call'.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents regression in cases where the coverage is not included.

Key Assertions:

- The `coverage` key in the result dictionary should contain exactly one entry.
- The `file_path` of the first `CoverageEntry` in the `coverage` list should be 'src/foo.py'.
- All other `CoverageEntry`s in the `coverage` list should have a different `file_path`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 case 'test_result_with_llm_opt_out' verifies that the TestCaseResult includes a flag indicating LLM opt-out.

Why Needed: This test prevents regression where the result does not include the flag for LLM opt-out, potentially causing incorrect interpretation of the output.

Key Assertions:

- The 'llm_opt_out' key in the result dictionary should be set to True.
- The value of the 'llm_opt_out' key should be a boolean indicating whether LLM opt-out is enabled or not.
- The TestCaseResult object should have been created with an LLM opt-out flag set to True.
- The 'llm_opt_out' attribute in the result dictionary should contain the correct boolean value.
- The 'llm_opt_out' key should be present in the result dictionary and its value should match the expected boolean value.
- The TestCaseResult object should have been created with an LLM opt-out flag set to True before calling the test function.
- The 'llm_opt_out' attribute in the result dictionary should contain the correct boolean value after calling the test 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/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 the `rerun_count` and `final_outcome` fields.

Why Needed: This test prevents regression where a rerun is not included in the result, potentially causing incorrect reporting of test results.

Key Assertions:

- The `rerun_count` field should be equal to 2.
- The `final_outcome` field should be equal to '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 include 'rerun_count' and 'final_outcome' keys.

Why Needed: This test prevents regression where a result is rerunned, causing missing field errors in the output.

Key Assertions:

- The 'rerun_count' key should be absent from the `result` dictionary.
- The 'final_outcome' key should be absent from the `result` dictionary.
- The presence of 'rerun_count' and 'final_outcome' keys in the `result` dictionary should be checked for missing fields.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that default values are set correctly for the TestConfig class.

Why Needed: This test prevents a potential regression where the default values of the TestConfig class may not be set correctly, potentially leading to unexpected behavior or errors.

Key Assertions:

- The cfg.provider should be set to 'none'.
- The cfg.llm_context_mode should be set to 'minimal'.
- The cfg.llm_max_tests should be set to 0.
- The cfg.llm_max_retries should be set to 3.
- The cfg.llm_context_bytes should be set to 32000.
- The cfg.llm_context_file_limit should be set to 10.
- The cfg.llm_requests_per_minute should be set to 5.
- The cfg.llm_timeout_seconds should be set to 30.
- The cfg.llm_cache_ttl_seconds should be set to 86400.
- The cfg.include_phase should be set to 'run'.
- The cfg.aggregate_policy should be set to 'latest'.
- The cfg.is_llm_enabled() should return False.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `get_default_config` function returns a default configuration with no provider.

Why Needed: Prevents a potential bug where the factory function does not return a valid configuration when no provider is specified.

Key Assertions:

- The returned value is an instance of `Config`.
- The `provider` attribute of the returned value is set to `''none''`.
- The `provider` attribute is not set to any other default value (e.g., 'default', 'api') when no provider is specified.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 whether the LLM is enabled based on the provider.

Why Needed: Prevents a potential bug where the LLM is enabled by default and incorrectly assumed to be enabled for certain providers.

Key Assertions:

- The `is_llm_enabled()` method returns False when the provider is 'none'.
- The `is_llm_enabled()` method should return True when the provider is 'ollama'.
- The `is_llm_enabled()` method correctly handles cases where the provider is not specified (i.e., 'none').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 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: Testing the `validate` method with an invalid include phase.

Why Needed: Prevents a potential bug where an invalid include phase is not properly validated and causes unexpected behavior.

Key Assertions:

- The `validate` method should return at least one error message for an invalid include phase.
- The error message should contain the invalid include phase 'lunch_break'.
- The test should fail when an invalid include phase is passed to the `include_phase` parameter.

COVERAGE

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

Key Assertions:

- The configuration object is created with an invalid provider.
- The validate method returns exactly one error.
- The error message contains the string 'Invalid provider '
- The error message does not contain any other relevant information.
- The error message does not indicate a specific problem with the configuration.
- The error message does not provide enough context to identify the issue.
- The error message is too vague and does not clearly state the problem.
- The test fails if an invalid provider is used.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test validation of numeric constraints for Config object.

Why Needed: Prevents a potential bug where the `llm_context_bytes` constraint is not enforced correctly, potentially leading to unexpected behavior or errors.

Key Assertions:

- cfg.validate() should return an error message indicating that llm_context_bytes must be at least 1000
- cfg.validate() should include 'llm_max_tests' in the error messages
- cfg.validate() should include 'llm_requests_per_minute' in the error messages
- cfg.validate() should include 'llm_timeout_seconds' in the error messages
- cfg.validate() should include 'llm_max_retries' in the error messages

COVERAGE

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

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

AI ASSESSMENT

Scenario: Valid configuration is validated without any issues.

Why Needed: A valid configuration is necessary to prevent potential bugs or regressions in the application.

Key Assertions:

- The Config object is created successfully.
- The validate() method returns an empty list of errors.
- No validation errors are reported for a well-formed configuration.
- The configuration is correctly validated without any invalid values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 test_load_aggregation_options function to ensure it correctly loads aggregation options.

Why Needed: To prevent a bug where the aggregation policy is not being loaded correctly, which could lead to unexpected behavior in the test suite.

Key Assertions:

- The aggregate_dir attribute of the configuration object should be set to 'aggr_dir'.
- The aggregate_policy attribute of the configuration object should be set to 'merge'.
- The aggregate_run_id attribute of the configuration object should be set to 'run-123'.
- The aggregate_group_id attribute of the configuration object should be set to 'group-abc'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	28 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286-294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test the test_load_config_invalid_int_ini function with invalid integer values in INI.

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

Key Assertions:

- The function should not crash or return unexpected results when it encounters an invalid integer value in the INI file.
- The default value of llm_max_retries is correctly set to 3, as intended.
- The mock_pytest_config.getini.side_effect is called with the correct argument (ini_values.get(key))
- The test asserts that cfg.llm_max_retries equals 3 after calling load_config(mock_pytest_config)
- The function returns a valid configuration object even when it encounters an invalid integer value 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: The test verifies that the `llm_coverage_source` option is correctly set to 'cov_dir' when loaded.

Why Needed: This test prevents a bug where the coverage source option is not properly configured, potentially leading to incorrect coverage metrics.

Key Assertions:

- cfg.llm_coverage_source
- cfg.llm_coverage_source == 'cov_dir'
- mock_pytest_config.option.llm_coverage_source
- mock_pytest_config.option.llm_coverage_source == 'cov_dir'
- cfg.llm_coverage_source is not set to an empty string or None.
- cfg.llm_coverage_source is not equal to 'default' when loaded from a configuration 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	25 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the default provider is not set or report HTML is not loaded.

Key Assertions:

- cfg.provider == 'none'
- cfg.report_html is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	24 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that CLI options override ini options.

Why Needed: This test prevents a potential bug where the CLI overrides ini settings, potentially causing unexpected behavior or errors.

Key Assertions:

- Set ini values correctly.
- Set CLI values correctly and override ini values.
- Verify correct report HTML value.
- Verify correct requests per minute value.
- Verify that CLI values are not overridden by ini values in this case.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 set, causing the `load_config` function to return an incorrect value.

Key Assertions:

- The `llm_max_retries` option should be set to 9 when loading configuration from CLI.
- The `load_config` function should raise a `ValueError` if the `llm_max_retries` option is not set to 9.
- The `load_config` function should correctly return an instance of `PytestConfig` with the correct `llm_max_retries` value when loading configuration from CLI.

COVERAGE

src/pytest_llm_report/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 'llm_report_provider' option is not set in the ini file.

Key Assertions:

- The 'provider' key should be set to 'ollama'.
- The 'model' key should be set to 'llama3'.
- The 'context_mode' key should be set to 'balanced'.
- The 'requests_per_minute' key should be set to 10.
- The 'max_retries' key should be set to 5.
- The 'report_html' key should be set to 'report.html'.
- The 'report_json' key should be set to 'report.json'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	32 lines (ranges: 107, 147, 248, 251-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test the configuration of aggregation settings.

Why Needed: Prevents regression in aggregation settings configuration.

Key Assertions:

- The aggregate_dir attribute should be set to '/reports'.
- The aggregate_policy attribute should be set to 'merge'.
- The aggregate_include_history attribute should be set to True.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test Config with all output paths.

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

Key Assertions:

- The `report_html` 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 `capture_failed_output` is set to `True` in the test configuration.

Why Needed: This test prevents a potential bug where the test fails due to an incorrect capture output limit.

Key Assertions:

- config.capture_failed_output is set to `True`
- config.capture_output_max_chars is not set to a valid value (8000)
- assertion error: expected 'None' but got a boolean value for `capture_failed_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	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 `Config` object is correctly initialized with compliance settings.

Why Needed: This test prevents a potential bug where the `Config` object's metadata file and HMAC key file are not set correctly.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Tests the configuration of coverage settings.

Why Needed: This test prevents a bug where the coverage settings are not correctly configured, potentially leading to incorrect coverage reports.

Key Assertions:

- config.omit_tests_from_coverage is False
- config.include_phase == "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 the inclusion of custom exclude globs in the configuration.

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

Key Assertions:

- The function `Config(llm_context_exclude_globs)` is called with the provided arguments.
- The variable `llm_context_exclude_globs` is set to `['*.pyc', '*.log']` in the test.
- The string `*.pyc` is found within the list of exclude globs.
- The string `*.log` is found within the list of exclude globs.
- The variable `config` is an instance of `Config` with the specified arguments.
- The method `llm_context_exclude_globs` is called on the `config` instance.
- The value of `llm_context_exclude_globs` is a list containing both `*.pyc` and `*.log` strings.

COVERAGE

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

Why Needed: This test prevents a potential bug where the `include_globs` configuration option is not properly propagated to the LLM context.

Key Assertions:

- The `*.py` glob matches one or more `.py` files in the current directory.
- The `*.pyi` glob matches one or more `.pyi` files in the current directory.
- The `include_globs` configuration option includes all specified Python files in the LLM context.
- The `llm_context_include_globs` attribute of the `Config` object contains a list of matched globs.
- The `include_globs` value is set to a list containing both `.py` and `.pyi` glob patterns.
- The `include_globs` configuration option is properly propagated to the LLM context, including all specified Python files.

COVERAGE

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

Why Needed: This test prevents a potential bug where the `include_pytest_invocation` option is incorrectly set to True, causing unexpected behavior in tests that rely on this feature.

Key Assertions:

- config.include_pytest_invocation is set to False
- config.include_pytest_invocation should be set to False based on the provided configuration options

COVERAGE

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

Why Needed: Prevents regression in LLM execution settings when using LLMS.

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 configuration does not exceed the maximum allowed LLMS requests per minute.
- The configuration does not exceed the maximum allowed LLMS concurrency.
- The configuration does not exceed the maximum allowed LLMS timeout in seconds.
- The configuration has a valid cache directory.
- The cache directory is set to a valid path.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Tests the configuration of LLM parameter settings.

Why Needed: This test prevents regression in LLM parameter setting configurations.

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: Verify the correct provider, model, and context bytes for LLM settings.

Why Needed: This test prevents a regression where the LLM settings are not properly configured due to incorrect values.

Key Assertions:

- The `provider` attribute is set to `ollama`.
- The `model` attribute is set to `llama3.2`.
- The `llm_context_bytes` attribute is set to `64000` bytes.
- The `llm_context_file_limit` attribute is not exceeded.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `repo_root` attribute of the `Config` object is correctly set to `/project`.

Why Needed: This test prevents a potential bug where the `repo_root` attribute is not set correctly, causing issues with file system operations.

Key Assertions:

- config.repo_root is an instance of `path.Path` and its value is equal to `/project`
- config.repo_root is correctly initialized before use
- the `repo_root` attribute is set to the correct path `/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 cause the test to fail.

Key Assertions:

- Verify that no errors are present in the configuration for each phase (run, setup, teardown, and all).
- Check if any error messages contain the string 'include_phase'.
- Confirm that the validation process correctly identifies invalid or missing include_phase values.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the default exclude globs for a minimal LLM configuration.

Why Needed: This test prevents potential issues with sensitive files being included in the model's training data.

Key Assertions:

- The function `*.pyc` is present in the defaults.
- The function `__pycache__/*` is present in the defaults.
- The function `*secret*` is present in the defaults.
- The function `*password*` is present in the defaults.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that default redact patterns include sensitive information.

Why Needed: This test prevents a potential security vulnerability where sensitive information is not properly redacted in the default configuration.

Key Assertions:

- The `--password` pattern should match any string containing `--password`.
- The `--token` pattern should match any string containing `--token`.
- The `--api[_-]?key` pattern should match any string containing `--api[_-]?key` (note: the `_-` is a special character in Python that matches any single character).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Verify default values of the test configuration.

Why Needed: Prevents regression in default configuration settings.

Key Assertions:

- The provider should be set to 'none'.
- The llm_context_mode should be set to 'minimal'.
- The llm_context_bytes should be set to 32000 bytes.
- The omit_tests_from_coverage flag should be True.
- The include_phase should be set to 'run'.

COVERAGE

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

PASSED `tests/test_options_maximal.py::TestConfigHelpersMaximal::test_is_llm_enabled` 1ms 3

AI ASSESSMENT

Scenario: Tests whether the `is_llm_enabled` method returns correct enabled status for different providers.

Why Needed: Prevents regression in LLM provider selection, ensuring that the method behaves as expected for various scenarios.

Key Assertions:

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

COVERAGE

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

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_aggregate_policy` 1ms 3

AI ASSESSMENT

Scenario: Testing the `validate` method of the `Config` class to ensure it correctly identifies and reports invalid aggregate policies.

Why Needed: To prevent a potential bug where an invalid aggregate policy is not properly reported, allowing for incorrect configuration or unexpected behavior.

Key Assertions:

- The `validate` method should return exactly one error message containing the string 'Invalid aggregate_policy 'invalid'.
- The error message should include the exact value of the invalid aggregate policy ('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, 193-197, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_context_mode` 1ms 3

AI ASSESSMENT

Scenario: Test validates the `Config` class with an invalid context mode.

Why Needed: Prevents a potential bug where the `validate()` method returns more than one error for an invalid context mode.

Key Assertions:

- The `config.validate()` method should return exactly one error message.
- The error message should contain 'Invalid llm_context_mode 'invalid''.
- The error message should be present in the first error object.
- The error message should not be empty.
- The context mode is invalid according to the LLM model configuration.

COVERAGE

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

Why Needed: Prevents a potential bug where the test fails due to an incorrect or missing error message for an invalid include phase.

Key Assertions:

- The function `validate()` should return exactly one error message.
- The error message should contain 'Invalid include_phase 'invalid''.
- The error message should be present in the first error found.
- The test should fail if only one error is returned from the validation process.
- The error message should not be empty or null.
- The error message should indicate that an invalid include phase was provided.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 bug where the test fails due to an invalid provider being used.

Key Assertions:

- The configuration should validate against an invalid provider.
- The error message for the invalid provider should be 'Invalid provider 'invalid'.
- The test should fail when using an invalid provider.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_numeric_bounds` 1ms 3

AI ASSESSMENT

Scenario:
tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_numeric_bounds

Why Needed: Prevents regression where invalid numeric values are not properly validated.

Key Assertions:

- The function should return a list of errors for invalid numeric values.
- The function should check if 'llm_context_bytes' is in the list of errors.
- The function should check if 'llm_max_tests' is in the list of errors.
- The function should check if 'llm_requests_per_minute' is in the list of errors.
- The function should check if 'llm_timeout_seconds' is in the list of errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	21 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209-217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_valid_config 1ms 3

AI ASSESSMENT

Scenario: Verifies that the `validate` method returns an empty list for a valid configuration.

Why Needed: Prevents a potential bug where an invalid configuration is passed to the validation process and causes unexpected behavior.

Key Assertions:

- The `validate` method should return an empty list for a valid configuration.
- An empty list should be returned when the input configuration is valid.
- The validation process should not crash or throw any exceptions when given a valid configuration.
- The test should fail with an assertion error if an invalid configuration is passed to the `validate` method.
- A valid configuration should not prevent the `validate` method from returning an empty list.
- The `validate` method's return value should be consistent across different test runs.
- The test should only fail when a valid configuration is provided, and not when other invalid configurations are used.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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` class has default settings when no configuration is provided.

Why Needed: This test prevents a potential bug where the plugin's configuration defaults to unexpected values if no configuration is specified.

Key Assertions:

- The `Config` instance should be an instance of `Config`.
- The `cfg` variable should have an attribute named `safe_defaults` and its value should be `True`.
- The `cfg.safe_defaults` attribute should return `True`.
- The `cfg.safe_defaults` attribute should not raise a `TypeError` when accessed directly.
- The `cfg.safe_defaults` attribute should not raise a `TypeError` when called with no arguments.
- The `safe_defaults` attribute of the `Config` instance should be `True`.
- The `safe_defaults` attribute of the `Config` instance should have an attribute named `default_config` and its value should be `None` or a string.
- The `default_config` attribute of the `Config` instance should not raise a `TypeError` when accessed directly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 prevents a potential bug where the plugin configuration is not available in the test setup.

Key Assertions:

- The `pytestconfig` object should be an instance of `pytest.config.Config` or its subclass.
- The `pytestconfig` object should have a `name` attribute.
- The `pytestconfig` object should have a `version` attribute.
- The `pytestconfig` object should have a `groups` attribute.
- The `pytestconfig` object should have a `extra_args` attribute.
- The `pytestconfig` object should be an instance of `Config` from the `pytest.config` module.
- The `pytestconfig` object should not be `None` when accessed.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the LLM context marker does not cause errors in a test.

Why Needed: The LLM context marker prevents tests from failing due to unexpected behavior or errors caused by the LLM.

Key Assertions:

- Asserts that the `test_llm_context_marker` function does not raise any exceptions when run without errors.
- Verifies that the `test_llm_context_marker` function correctly handles cases where the LLM context marker is present but does not cause an error.
- Checks for any unexpected behavior or errors caused by the LLM context marker in the test.
- Ensures that the test can still be run successfully without raising any exceptions due to the LLM context marker.

COVERAGE

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

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: The `requirement_marker` function is tested to ensure it does not throw any errors when used.

Why Needed: This test prevents a potential bug where the `requirement_marker` function could be misused and cause unexpected behavior.

Key Assertions:

- The `requirement_marker` function should return an empty list or None if no requirements are present.
- The `requirement_marker` function should not raise any exceptions when called with a non-empty list of requirements.
- The `requirement_marker` function should correctly identify and exclude irrelevant requirements from the output.
- The `requirement_marker` function should handle edge cases where there are no requirements or only one requirement.
- The `requirement_marker` function should not modify the original input list in any way.
- The `requirement_marker` function should be able to handle large lists of requirements without performance issues.
- The `requirement_marker` function should correctly group related requirements together.
- The `requirement_marker` function should return a clear and concise output that makes sense for the context.

COVERAGE

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

PASSED `tests/test_plugin_integration.py::TestReportGeneration::test_report_writer_integration` 37ms 6

AI ASSESSMENT

Scenario: Verify the integration of report writer with pytest_llm_report, ensuring it generates a full report and verifies its contents.

Why Needed: This test prevents regression by verifying that the report writer correctly generates a full report with summary statistics.

Key Assertions:

- The 'report.json' file exists in the specified path.
- The total count of tests passed is 1 out of 2.
- The number of tests passed is correct based on the test results.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: To prevent a regression where collectreport does not skip when disabled due to an incorrect implementation of pytest_collectreport.

Key Assertions:

- The `pytest_collectreport` function should be called with `_enabled_key` set to `False` and the result should match the expected behavior.
- The `get` method on the session config stash should return `None` when `_enabled_key` is not found.
- The `session.config.stash.get` call should assert that it was called with `_enabled_key` as an argument and a boolean value of `False` as its result.
- The mock report object returned by `pytest_collectreport` should be a valid instance of the `MagicMock` class.
- The `session.config.stash.get` method on the mock report object should return `None` when `_enabled_key` is not found.
- The `get` method on the session config stash should raise an error if it cannot find `_enabled_key` in the stash.
- The `pytest_collectreport` function should be able to handle cases where `_enabled_key` is not present in the stash without raising an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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 `pytest_collectreport` calls `_collector_key` when collectreport is enabled.

Why Needed: This test prevents a potential regression where the collector does not call the stash for enabled collectreport.

Key Assertions:

- The function `mock_collector.handle_collection_report.assert_called_once_with(mock_report)` should be called with one argument, which is an instance of `pytest_collectreport`.
- The function `mock_collector.handle_collection_report.assert_called_once_with(mock_report)` should not raise any exception.
- The function `mock_collector.handle_collection_report.assert_called_once_with(mock_report)` should return a tuple containing the report.
- The function `stash_get(key, default=None)` should call `_collector_key` when it is called with key '_collector_key'.
- The function `stash_get(key, default=None)` should not call `_enabled_key` when it is called with key '_collector_key'.
- The function `stash_get(key, default=None)` should return the correct value for key '_collector_key' (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	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 skips when a Pytest session is not available.

Why Needed: Prevent regression in plugin behavior when a Pytest session is not set.

Key Assertions:

- The `session` attribute of the mock report object is not present.
- The `pytest_collectreport` function does not raise an exception when called with a non-existent session.
- The `pytest_collectreport` function skips the test run without raising any errors.
- The `pytest_collectreport` function does not throw a `AttributeError` when trying to access the `session` attribute.
- The `session` attribute is set to an empty dictionary or None in the mock report object.
- The `session` attribute is set to a different value than what was expected (e.g., a string instead of a boolean).
- The `pytest_collectreport` function does not raise an exception when called with a session that has already been cleaned up.

COVERAGE

`src/pytest_llm_report/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 exception when the session is `None`.

Why Needed: Prevent a potential bug where `pytest_collectreport` raises an error when the session is `None`.

Key Assertions:

- The function `pytest_collectreport` should not be called with a `None` argument for the `session` attribute.
- The `session` attribute of the mock report object should remain unchanged after setting it to `None`.
- No exception should be raised when calling `pytest_collectreport` with a `None` session.
- The `pytest_collectreport` function should not throw an error or crash when given a `None` session.
- The `session` attribute of the mock report object should still have its original value after setting it to `None`.
- The test case should be able to run without any issues due to the `None` session.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	8 lines (ranges: 387-388, 391, 395-397, 408, 412)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_llm_enabled_warning` 3ms 3

AI ASSESSMENT

Scenario: Test that LLM enabled warning is raised when configuring Pytest with the llm_report_provider set to 'ollama'.

Why Needed: This test prevents a regression where the LLM provider 'ollama' might be enabled by default without user intervention, potentially causing unexpected behavior or errors.

Key Assertions:

- The value of `llm_report_provider` is set to 'ollama'.
- The value of `llm_report_model` is set to 'llama3.2'.
- The value of `llm_report_context_mode` is set to 'minimal'.
- The value of `llm_report_html` is None.
- The value of `llm_report_json` is None.
- The value of `llm_report_pdf` is None.
- The value of `llm_evidence_bundle` is None.
- The value of `llm_dependency_snapshot` is None.
- The value of `llm_requests_per_minute` is None.
- The value of `llm_aggregate_dir` is None.
- The value of `llm_aggregate_policy` is None.
- The value of `llm_aggregate_run_id` is None.
- The value of `llm_aggregate_group_id` is None.
- The value of `llm_max_retries` 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	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 a more informative error message.

Key Assertions:

- mock_config.option.llm_report_html is None
- mock_config.option.llm_report_json is None
- mock_config.option.llm_report_pdf is None
- mock_config.option.llm_evidence_bundle is None
- mock_config.option.llm_dependency_snapshot is None
- mock_config.option.llm_requests_per_minute is None
- mock_config.option.llm_aggregate_dir is None
- mock_config.option.llm_aggregate_policy is None
- mock_config.option.llm_aggregate_run_id is None
- mock_config.option.llm_aggregate_group_id is None
- mock_config.option.llm_max_retries is None
- mock_config.rootpath is /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	43 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 248, 251-253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	25 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-199, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_worker_skip` 1ms 2

AI ASSESSMENT

Scenario: Test that configure skips on xdist workers and verifies the correct behavior.

Why Needed: This test prevents a potential regression where pytest_configure() might be called unnecessarily when configuring with xdist workers.

Key Assertions:

- The `addinivalue_line` method is not called before checking for worker input.
- The `addinivalue_line` method is called after verifying the worker input.
- The `addinivalue_line` method is not called when configuring without xdist workers.
- The `addinivalue_line` method is correctly called only when a marker is added to the test.
- The `workerinput` attribute of the mock configuration object is set correctly.
- The `workerid` attribute of the worker input dictionary matches the expected value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 fallback to load_config if Config.load is missing during Pytest configuration.

Why Needed: Prevents a potential bug where the plugin fails to configure due to missing `Config.load` method call.

Key Assertions:

- Mocking `pytest_llm_report.options.Config` with no `load()` method ensures it's called correctly when `Config.load` is missing.
- The `validate()` method of `mock_cfg` returns an empty list, indicating successful configuration.
- The `load_config()` method of `mock_load` is called once, ensuring the fallback to load_config occurs as expected.
- The `option.llm_report_html` and `option.llm_max_retries` are not set, preventing a potential regression in plugin 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/plugin.py	29 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_all_ini_options` 2ms 3

AI ASSESSMENT

Scenario: Test loading all INI options for the plugin.

Why Needed: This test prevents regression in case the CLI options are set but not loaded correctly from INI files.

Key Assertions:

- The provider is set to 'ollama' when the option is None.
- The model is set to 'llama3.2' when the option is None.
- The context mode is set to 'complete' when the option is None.
- The requests per minute is set to 10 when the option is None.
- The report HTML is set to 'ini.html' when the option is None.
- The report JSON is set to 'ini.json' when the option is None.
- The aggregate directory is not set when the option is None.
- The aggregate policy is not set when the option is None.
- The aggregate run ID is not set when the option is None.
- The aggregate group ID is not set when the option is None.
- The maximum retries are not set when the option is None.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`31 lines (ranges: 107, 147, 248, 251-263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_cli_overrides_ini` 2ms 3

AI ASSESSMENT

Scenario: Test CLI options override INI options.

Why Needed: This test prevents a bug where the plugin's configuration is not properly overridden from INI files when using the command line interface.

Key Assertions:

- The `llm_report_html` option should be set to 'cli.html'.
- The `llm_report_json` option should be set to 'cli.json'.
- The `llm_report_pdf` option should be set to 'cli.pdf'.
- The `llm_evidence_bundle` option should be set to 'bundle.zip'.
- The `llm_dependency_snapshot` option should be set to 'deps.json'.
- The `llm_requests_per_minute` option should be set to 20.
- The `aggregate_dir` option should be set to '/agg'.
- The `aggregate_policy` option should be set to 'merge'.
- The `aggregate_run_id` option should be set to 'run-123'.
- The `aggregate_group_id` option should be set to 'group-abc'.
- The root path of the configuration file should be '/project'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`38 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259-263, 270-283, 286-295, 298, 300)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::test_terminal_summary_disabled` 1ms 2

AI ASSESSMENT

Scenario: Test that terminal summary skips when plugin is disabled.

Why Needed: Prevents a regression where the test fails due to an uncaught assertion error in pytest_terminal_summary.

Key Assertions:

- mocked stash.get() with False value for _enabled_key
- assert_called_once_with(_enabled_key, False) on mock_config.stash.get

COVERAGE

src/pytest_llm_report/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.

Why Needed: This test prevents a regression where the plugin does not skip the terminal summary for an xdist worker.

Key Assertions:

- The `pytest_terminal_summary` function should return early without doing anything when it encounters an xdist worker.
- The `workerinput` attribute of the mock configuration object is set to 'gw0' as expected.
- The test result is None, indicating that the plugin skipped the terminal summary 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	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 it correctly sets the report HTML path.

Why Needed: This test prevents a potential bug where the report HTML path is not set correctly when using pytest's CLI and INI options.

Key Assertions:

- The `report_html` option of the `load_config` function is set to 'out.html' as expected.
- The `rootpath` option of the `load_config` function is set to '/root' as expected.
- The `getini` method of the `mock_config` object returns None for the `llm_report_html` key as expected.
- The `report_html` attribute of the loaded configuration object is 'out.html' as expected.
- The `rootpath` attribute of the loaded configuration object is '/root' 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	36 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270-283, 286-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_disabled` 2ms 2

AI ASSESSMENT

Scenario: Test makereport skips when disabled.

Why Needed: This test prevents a regression where makereport is disabled and the plugin does not generate reports.

Key Assertions:

- The `pytest_runtest_makereport` hookwrapper should be able to skip the generation of reports when `makereport` is disabled.
- The `get_result()` method of the mock_outcome object should return a StopIteration exception when called with a generator that yields point.
- The `send()` method of the mock_outcome object should not raise an exception when called with a generator that yields point.
- The `stash.get()` method of the mock_item object should return False when called with no arguments.
- The `get_result()` method of the mock_outcome object should be able to handle the case where the generator yields point without raising an exception.
- The `send()` method of the mock_outcome object should not raise an exception when called with a generator that yields point.

COVERAGE

`src/pytest_llm_report/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 collect and report test results even if makereport is enabled.

Key Assertions:

- The `pytest_runtest_makereport` function should be able to find the `mock_collector` instance.
- The `mock_collector.handle_runtest_logreport` method should be called with the correct arguments.
- The `mock_report` object returned by `stash_get(_collector_key)` should not be None.
- The `mock_item.config.stash.get(_enabled_key)` call should return True for `_enabled_key` and `mock_collector`.
- The `mock_call.send(mock_outcome)` call should yield a point after the `send` method is called on `mock_outcome`.
- The `pytest_runtest_makereport` function should not raise an exception when called with a mock outcome.
- The `mock_collector.handle_runtest_logreport` method should be able to handle the `mock_report` object 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)

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.

Why Needed: To prevent a regression where the plugin fails to collect data 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 have been called with _enabled_key as True before calling it.
- The pytest_collection_finish function should have been called with _enabled_key as False after setting stash.get to False in the mock session.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 387-388, 391, 395-397, 431-432)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_enabled` 2ms 2

AI ASSESSMENT

Scenario: Test that collection_finish is called when enabled and Pytest is collecting data.

Why Needed: This test prevents a potential regression in the plugin where Pytest collection finish is not triggered correctly when it's enabled.

Key Assertions:

- The `pytest_collection_finish` function should be called with the collected items.
- The `handle_collection_finish` method of the collector should be called once with the collected items.
- The `items` attribute of the session should contain a list of collected items.
- Each item in the collection should have an `__getitem__` method that returns the item.
- The `stash_get` function should return True for `_enabled_key` and `True` for `_collector_key` when called with the correct keys.
- The `handle_collection_finish` method of the collector should not be called if there are no collected items.
- The `items` attribute of the session should contain a list of collected items after calling `pytest_collection_finish`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	10 lines (ranges: 387-388, 391, 395-397, 431, 435-437)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_disabled` 1ms 2

AI ASSESSMENT

Scenario: Test that pytest_sessionstart skips when sessionstart is disabled.

Why Needed: This test prevents a regression where the plugin fails to check if the session has been started.

Key Assertions:

- mocked config.stash.get.call_count should be 1
- mocked config.stash.get._enabled_key should have been called with False as argument
- pytest_sessionstart was not called with mock_session
- mock_session.config.stash.get.return_value should have been set to False
- mock_session.config.stash.get.assert_called_with(_enabled_key, False) should have been called

COVERAGE

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

Key Assertions:

- The `_collector_key` should be present in the mock stash.
- The `_start_time_key` should be present in the mock stash.
- A `MockStash` instance with a valid stash dictionary is created and configured 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: This test prevents a potential bug where pytest_addoption 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 for llm_report plugin.

Why Needed: This test prevents a regression where the plugin does not add INI options when using pytest.

Key Assertions:

- Verify that 'llm_report_html' is present in the ini calls.
- Verify that 'llm_report_json' is present in the ini calls.
- Verify that 'llm_report_max_retries' is present in the ini calls.

COVERAGE

src/pytest_llm_report/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 `_enabled_key` and `_config_key` values.
- The `mock_cov_cls.return_value` method call sets up a mock Coverage instance.
- The `mock_cov.report.return_value` method call returns the expected coverage percentage (85.5%).
- The `pytest_terminal_summary` function is called with a valid configuration object.
- The `MockStash` class is used to simulate the stash dictionary.
- The `CoverageMapper` and `ReportWriter` patches are used to mock the Coverage and ReportWriter classes respectively.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	58 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-305, 307, 309-315, 317-318, 331-332, 337-338, 365-375, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_llm_enabled` 3ms 3

AI ASSESSMENT

Scenario: Test terminal summary with LLM enabled runs annotations.

Why Needed: Prevents regression in terminal summary functionality when LLM is enabled.

Key Assertions:

- Verify that the `pytest_terminal_summary` function is called with the correct configuration.
- Assert that the `cfg` variable contains a boolean value of True for `_enabled_key` and a valid configuration object for `_config_key`.
- Check if the `mock_config.stash` dictionary has been correctly populated with mock data.
- Verify that the `mock_annotate.call_args[0][1]` attribute is set to the provided configuration object.
- Assert that the `mock_annotate.call_args[0][1]` attribute contains a valid model name for the LLM provider.
- Check if the `pytest_terminal_summary` function is patched correctly at source.
- Verify that the mock dependencies are properly patched and their methods are called as expected.
- Assert that the `mock_writer_cls.return_value` attribute is set to the correct mock writer instance.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	59 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331-332, 337-340, 343, 345, 348-350, 357-362, 365-375, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_no_collector` 2ms 3

AI ASSESSMENT

Scenario: Test terminal summary creates collector if missing.

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

Key Assertions:

- mock_terminalreporter.call_args_list[0][1] == True
- stash._enabled_key == True
- stash._config_key == cfg
- mock_config.stash._enabled_key == False
- mock_config.stash._config_key == cfg
- mock_writer_cls.return_value.__class__.__name__ == 'MockWriter'
- mock_mapper_cls.return_value.__class__.__name__ == 'MockCoverageMapper'

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 a regression where the aggregation feature is disabled and the plugin does not produce any meaningful output.

Key Assertions:

- The `aggregate_dir` parameter in the configuration should be set to `/agg` when aggregation is enabled.
- The `get()` method of the stash should return an empty list when no data is available.
- The `[]` indexing should not raise an error when trying to access a non-existent key in the stash.
- The `aggregate()` method of the aggregator should be called once with a valid report object.
- The `ReportWriter` instance should have been written to JSON and HTML files.
- The `ReportWriter` instance's `write_json()` and `write_html()` methods should have been called 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	21 lines (ranges: 238, 242, 246, 249-250, 252-253, 256-257, 259, 261-265, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummaryErrors::test_terminal_summary_coverage_error` 4ms 3

AI ASSESSMENT

Scenario: Test coverage calculation error when loading a large coverage map.

Why Needed: This test prevents regression where the plugin fails to calculate coverage due to an OSError during load.

Key Assertions:

- The `load` method of `CoverageMapper` should not raise an exception if the coverage map is empty.
- The `coverage_map` attribute of `CoverageMapper` should be a list or tuple instead of a single value.
- The `report_writer` attribute of `ReportWriter` should be set to `None` when no report writer is configured.
- The `pytest_terminal_summary` function should not raise an exception if the coverage percentage is zero.
- The `report_html` option should be set to `''out.html''` instead of a file path.
- The `llm_coverage_source` option should be set to `.coverage` instead of a file path.
- The `stash` attribute of `Config` should contain the correct values when the plugin is enabled and coverage is enabled.
- The `workerinput` attribute of `Config` should be deleted before the test runs.

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` 7ms 4

AI ASSESSMENT

Scenario: Test the ContextAssembler to assemble a balanced context for a test file.

Why Needed: This test prevents regression when using an unbalanced llm_context_mode, as it ensures that all required dependencies are present in the assembly.

Key Assertions:

- The 'utils.py' file is included in the assembled context.
- The 'def util()' function is found in the 'utils.py' file within the assembled context.
- All required dependencies (in this case, 'util') are present 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: Test that the ContextAssembler can assemble a complete context for a test file with no imports.

Why Needed: This test prevents a potential bug where the ContextAssembler fails to assemble a complete context for a test file with no imports, resulting in an assertion error.

Key Assertions:

- The `source` variable contains the assembled source code of the test file.
- The `context` variable is not empty and does not contain any import statements.
- The `source` variable contains the expected test function definition 'test_1'.
- The `context` variable is a valid context object with no imports.
- The `source` variable has the correct number of lines (3) and the correct indentation.
- The `source` variable does not contain any import statements from other files.
- The `context` variable contains the expected test function definition 'test_1' in the correct location.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/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: Assembling a minimal context for testing test_1 in test_a.py

Why Needed: Prevents regression when minimal context is not used by the llm.

Key Assertions:

- The 'test_1' function should be found in the source code of test_a.py.
- The assembly result should have a nodeid of 'test_a.py::test_1'.
- The context should be an empty dictionary when assembling the minimal context.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	30 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 116)

PASSED `tests/test_prompts.py::TestContextAssembler::test_balanced_context_limits` 1ms 4

AI ASSESSMENT

Scenario: Verify that the ContextAssembler does not exceed the specified context limits when assembling a test file.

Why Needed: This test prevents a potential bug where the ContextAssembler exceeds the specified context limit, causing unexpected behavior or errors.

Key Assertions:

- The assembler should be able to assemble the test file without truncating any part of it.
- The assembled context should contain all original content from the test file.
- The length of the assembled context should not exceed the specified limit (40 bytes in this case).
- Any truncated content should be clearly indicated within the assembled context.
- The assembler should handle files with varying lengths without truncating them.
- The assembler should preserve the original line count and coverage information from the test file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	34 lines (ranges: 33, 49, 52, 55, 58, 60-61, 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 that the ContextAssembler correctly handles non-existent files and nested test names with parameters.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly returns an empty string for a file that does not exist or has a nested test name with parameters.

Key Assertions:

- assert 'def test_param' in source
- assert 'test_p.py::test_param[1]' in 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/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 sensitive files in the LLM context.

Key Assertions:

- asserts that 'test.pyc' is excluded because it's a Python bytecode file and should be excluded by default.
- asserts that 'secret/key.txt' is excluded because it contains sensitive information that should not be included in the LLM context.
- asserts that 'public/readme.md' is excluded because it does not contain any sensitive information that would require its inclusion in 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)
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: Consecutive lines in the input list should be compressed to a single range.

Why Needed: This test prevents regression when consecutive lines are present in the input data.

Key Assertions:

- The function `compress_ranges([1, 2, 3])` returns the correct range for consecutive lines (e.g., '1-3'),
- the function should handle cases where there are no consecutive lines or multiple non-consecutive lines in the input list.
- the function should correctly handle edge cases such as empty lists or lists with only one element.
- the function should return an error message for invalid input data (e.g., a list containing non-integer values).
- the function should preserve the original order of elements within each range.
- the function should be able to handle ranges that span multiple lines in the input data.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)`

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

AI ASSESSMENT

Scenario: The test verifies that the function correctly handles duplicate ranges.

Why Needed: This test prevents a potential bug where the function may incorrectly identify or handle duplicate ranges as separate.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

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

AI ASSESSMENT

Scenario: Testing the `compress_ranges` function with an empty input.

Why Needed: This test prevents a potential bug where an empty list is returned, causing unexpected behavior in downstream code.

Key Assertions:

- The `compress_ranges` function should return an empty string for an empty input.
- The `compress_ranges` function should not raise any exceptions when given an empty input.
- The `compress_ranges` function should maintain its original behavior when given a non-empty list as input.
- The `compress_ranges` function should handle the case where the input is a single-element list correctly.
- The `compress_ranges` function should be able to identify and return the correct range for an empty list.
- The `compress_ranges` function should not produce any unexpected output when given an empty list as input.
- The `compress_ranges` function should maintain its original order of operations when given a non-empty list 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/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 `compress_ranges` with a mixed range of values.

Why Needed: The test prevents regression in cases where ranges are mixed with single values.

Key Assertions:

- The output should be '1-3, 5, 10-12, 15' as per the expected format.
- Each range value should be correctly separated by a comma.
- Single values within ranges should not affect the overall output.
- No single values should appear in the output without being part of a range.
- The function should handle cases where there are no ranges (i.e., all values are singles).
- The function should still produce the expected output even if the input list contains duplicate values within ranges.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)

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

AI ASSESSMENT

Scenario: Non-consecutive lines in the input list are expected to be comma-separated.

Why Needed: This test prevents regression when non-consecutive lines are present in the input data.

Key Assertions:

- The function should return a string with comma-separated values for non-consecutive lines.
- The function should handle cases where there is only one value in the list.
- Non-consecutive lines should be separated by commas, not spaces or other characters.
- The function should ignore leading zeros and trailing spaces when compressing the input list.
- Non-consecutive lines with different lengths should also be comma-separated.
- Leading zeros should be ignored when comparing compressed lists to the expected output.
- Trailing spaces should be ignored when comparing compressed lists to the expected output.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	14 lines (ranges: 29, 33, 35-37, 39-40, 45-47, 50, 52, 65-66)

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

AI ASSESSMENT

Scenario: tests/test_ranges.py::TestCompressRanges::test_single_line

Why Needed: This test prevents a potential bug where the single-line function would incorrectly compress ranges.

Key Assertions:

- The input list should contain only one element.
- The compressed range should be equal to the original range.
- No range notation should be used in the output string.
- The output string should not contain any extraneous characters or whitespace.
- The function should handle empty lists correctly.
- The function should return an error message for invalid input (e.g. non-numeric values).
- The function should preserve the original order of elements in the 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	10 lines (ranges: 29, 33, 35-37, 39, 50, 52, 65-66)

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

AI ASSESSMENT

Scenario: Testing the `compress_ranges` function with two consecutive line numbers.

Why Needed: Prevents regression where two consecutive line numbers are compressed to a single number.

Key Assertions:

- The input list contains at least two elements.
- At least one element in the input list is an integer.
- All elements in the input list are integers.
- The function correctly compresses two consecutive line numbers.
- The compressed range does not contain any non-integer values.
- The function handles edge cases where the input list contains only one or zero 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: Testing the `compress_ranges` function with an unsorted input.

Why Needed: This test prevents a potential bug where the function incorrectly handles unsorted ranges.

Key Assertions:

- The function should correctly compress the ranges and return a string in the format 'start-end',
- e.g. '1-3' for the range [5, 1, 3].
- or '2-4' for the range [2, 4] if the input is already sorted.
- The function should handle duplicate ranges correctly and not produce any errors.
- e.g. ['1-3', '2-4'] for the input [5, 1, 3, 2].
- or ['2-4', '1-3'] for the input [2, 4] if the input is already sorted.
- The function should return an empty string if the input is empty.
- e.g. '' for the input [].
- or 'None' for the input [None, None, None].
- The function should handle non-integer values correctly and not produce any errors.
- e.g. ['1', '2'] for the input [5, 1, 3, 2] with non-integer values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 regression where the function returns incorrect results for empty input strings.

Key Assertions:

- Input: An empty string
- Expected output: An empty list
- Check if the function correctly handles empty strings by comparing its result with an empty list
- Verify that the function does not throw any errors when given an empty string as input
- Check for any potential edge cases, such as a single-character string or a string containing only whitespace characters

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/ranges.py`	`2 lines (ranges: 81-82)`

PASSED tests/test_ranges.py::TestExpandRanges::test_mixed 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `expand_ranges` function correctly handles mixed ranges and singles.

Why Needed: This test prevents a potential bug where the function incorrectly treats single numbers as if they were ranges.

Key Assertions:

- The function should split the string into individual elements (1, 5, 10-12) and convert them to integers before returning a list.
- The function should correctly handle single numbers in the range (1, 3).
- The function should not treat single numbers as if they were ranges (e.g., 2 is considered a range of [1, 3]).

COVERAGE

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

Why Needed: This test prevents the function from expanding the range incorrectly, potentially leading to incorrect results or errors in downstream code.

Key Assertions:

- The input string should be in the format 'start-end' (e.g., '1-3')
- The start value should be less than or equal to the end value
- All values between the start and end should be included in the output 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`	10 lines (ranges: 81, 84-91, 95)

PASSED `tests/test_ranges.py::TestExpandRanges::test_roundtrip` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that `compress_ranges` and `expand_ranges` are inverses by comparing the original list with its roundtrip expansion.

Why Needed: This test prevents a potential bug where the order of elements in the compressed or expanded range is not preserved, leading to incorrect results.

Key Assertions:

- The original list `[1, 2, 3, 5, 10, 11, 12, 15]` should be equal to its roundtrip expansion `original`.
- The compressed range `[1, 2, 3, 5, 10]` should be equal to the expanded range `[1, 2, 3, 5, 10, 11, 12, 15]`.
- All elements in the original list should be present in the roundtrip expansion.
- No element outside the original list should be present in the roundtrip expansion.
- The order of elements within each range should be preserved.
- No duplicate ranges should be created during the roundtrip process.
- All ranges should have a unique start and 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	27 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67, 81, 84-91, 93, 95)

PASSED tests/test_ranges.py::TestExpandRanges::test_single_number 1ms 3

AI ASSESSMENT

Scenario: The 'expand_ranges' function is expected to handle a single input number correctly.

Why Needed: This test prevents the function from producing an incorrect result for a single number, potentially leading to unexpected behavior or errors in downstream calculations.

Key Assertions:

- Input: '5'
- Expected output: [5]
- Expansion of ranges should not be necessary
- Single input number does not require expansion
- No need to check if the input is a valid number
- The function should return an empty list for single numbers

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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 function formats duration in milliseconds for values less than 1 second.

Why Needed: Prevents a potential bug where the function does not correctly format durations below 1 second as '500ms' instead of '1ms'.

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 that the duration function correctly formats seconds for values greater than or equal to 1 second.

Why Needed: This test prevents a potential bug where the function does not handle cases where the input is less than 1 second correctly.

Key Assertions:

- The output of format_duration(1.23) should be '1.23s'.
- The output of format_duration(60.0) should be '60.00s'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/render.py`	`2 lines (ranges: 65-66)`

PASSED tests/test_render.py::TestOutcomeToCssClass::test_all_outcomes 1ms 3

AI ASSESSMENT

Scenario: All outcomes should map to CSS classes.

Why Needed: Prevents regression in CSS class mapping for different outcome types.

Key Assertions:

- The function `outcome_to_css_class` correctly maps all outcome strings to their corresponding CSS classes.
- It handles the special case of 'xfailed' by returning 'outcome-xfailed'.
- For outcomes like 'passed', it returns 'outcome-passed'.
- The function also correctly maps 'error' to '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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	8 lines (ranges: 79-85, 87)

PASSED tests/test_render.py::TestOutcomeToCssClass::test_unknown_outcome 1ms 3

AI ASSESSMENT

Scenario: Tests the 'outcome_to_css_class' function with an unknown outcome.

Why Needed: This test prevents a potential bug where the function returns incorrect CSS classes for unknown outcomes.

Key Assertions:

- The function should return 'outcome-unknown' when given an unknown outcome.
- The function should not return any other class (e.g. 'outcome-error', 'outcome-success')
- The function should handle cases where the outcome is not recognized by returning a default CSS class ('outcome-unknown').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	8 lines (ranges: 79-85, 87)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_basic_report` 1ms 3

AI ASSESSMENT

Scenario: Tests the rendering of a basic report with fallback HTML.

Why Needed: This test prevents regression where the rendered report contains incomplete or incorrect information due to missing or malformed HTML.

Key Assertions:

- The '' header is present in the rendered HTML.
- The 'Test Report' title is displayed in the rendered HTML.
- The 'test::passed' and 'test::failed' nodeids are found in the rendered HTML.
- The 'PASSED' and 'FAILED' labels are displayed in the rendered HTML.
- The 'Plugin: v0.1.0' and 'Repo: v1.2.3' text is present in the rendered HTML.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	52 lines (ranges: 65-67, 79-85, 87, 121-124, 126-127, 131-132, 141-143, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_coverage` 1ms 3

AI ASSESSMENT

Scenario: Test verifies that the test renders coverage information.

Why Needed: This test prevents regression and ensures that the test renders coverage information correctly.

Key Assertions:

- The report root contains a 'Summary' object with a total of 1 test and 1 passed test.
- The 'CoverageEntry' object in the 'tests' list has a file path of 'src/foo.py', indicating that it covers lines 1-5.
- The 'html' variable is set to include the source code of 'src/foo.py'.
- The assertion checks if the string '5 lines' is present in the rendered HTML. If not, it will fail and trigger a test failure.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	52 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-129, 131-132, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_llm_annotation` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the LLM annotation is included in the rendered HTML report.

Why Needed: This test prevents a regression where the LLM annotation is not displayed correctly, potentially allowing for authentication bypass.

Key Assertions:

- The string "Tests login flow" should be present in the rendered HTML report.
- The string "Prevents auth bypass" should be present in the rendered HTML report.
- The LLM annotation nodeid should match with "test::foo" in the test result.
- The outcome of the test should be 'passed' according to the summary.
- The total number of tests passed should be 1.
- The number of tests that failed should be 0.
- The number of tests that were skipped should be 0.
- The number of tests with errors should be 0.
- The report root should have a valid end_time.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/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: Tests the inclusion of source coverage summary in the rendered HTML.

Why Needed: This test prevents a regression where the source coverage is not displayed correctly.

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 in the 'Source Coverage' section.
- The ranges of missed code ('1-4, 6-8') should be included in the 'Source Coverage' section.
- The ranges of missed code ('5, 9-10') should not be included 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' and 'XPassed' summary entries are not displayed correctly due to missing fallback HTML.

Key Assertions:

- The 'XFailed' summary entry should be present in the rendered HTML.
- The 'XPassed' summary entry should be present in the rendered HTML.
- Both 'XFailed' and 'XPassed' summary entries should be included in the rendered HTML, without any missing or incorrect 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/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	50 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-127, 131-132, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_report_writer.py::TestComputeSha256::test_different_content` 1ms 3

AI ASSESSMENT

Scenario: Test 'different_content' verifies that the same input produces different hashes.

Why Needed: This test prevents a potential bug where two inputs with the same content but different encoding produce the same output hash.

Key Assertions:

- hash1 != hash2
- assert isinstance(hash1, int) and isinstance(hash2, int)
- assert len(hash1) == len(hash2)
- assert all(isinstance(c, str) for c in hash1) and all(isinstance(c, str) for c in hash2)
- assert all(c in hash1 for c in 'abcdefghijklmnopqrstuvwxyz') and all(c in hash2 for c in 'abcdefghijklmnopqrstuvwxyz')
- assert all(c in hash1 for c in '0123456789abcdefABCDEF') and all(c in hash2 for c in '0123456789abcdefABCDEF')
- assert len(hash1) == 32 and len(hash2) == 32

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

PASSED tests/test_report_writer.py::TestComputeSha256::test_empty_bytes 1ms 3

AI ASSESSMENT

Scenario: Test 'test_empty_bytes': Verifies that an empty byte string produces consistent hash.

Why Needed: To prevent a potential bug where the test fails due to incorrect expected hash values for empty byte strings.

Key Assertions:

- The function `compute_sha256(b'')` should produce a consistent hash value.
- The length of the resulting hash value should be equal to 64 bytes (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` 9ms 4

AI ASSESSMENT

Scenario: Test the `build_run_meta` method of ReportWriter to ensure it includes version info.

Why Needed: This test prevents regression where the report writer does not include version information in the run metadata.

Key Assertions:

- The duration of the run should be 60 seconds.
- The pytest version should have a value.
- The plugin version should be '0.1.0'.
- The Python version should also be present.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	67 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300)

PASSED `tests/test_report_writer.py::TestReportWriter::test_build_summary_all_outcomes` 1ms 4

AI ASSESSMENT

Scenario: Test verifies that the `build_summary` method counts all outcome types correctly.

Why Needed: This test prevents a regression where the summary does not accurately count all outcome types, potentially leading to incorrect reporting of test results.

Key Assertions:

- asserts that the total count of outcomes is equal to 6 (all outcome types: passed, failed, skipped, xfailed, xpassed, error)
- asserts that the number of passed outcomes is equal to 1
- asserts that the number of failed outcomes is equal to 1
- asserts that the number of skipped outcomes is equal to 1
- asserts that the number of xfailed outcomes is equal to 1
- asserts that the number of xpassed outcomes is equal to 1
- asserts that the number of error outcomes is equal to 1

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	19 lines (ranges: 156-158, 312, 314-315, 317-328, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_build_summary_counts` 1ms 4

AI ASSESSMENT

Scenario: The test verifies that the `test_build_summary_counts` method correctly counts the total number of tests, passed tests, failed tests, and skipped tests.

Why Needed: This test prevents regression in cases where the output of the `_build_summary` method changes unexpectedly.

Key Assertions:

- total should be equal to 4 (number of tests)
- passed should be equal to 2 (number of passed tests)
- failed should be equal to 1 (number of failed tests)
- skipped should be equal to 1 (number of skipped tests)

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/report_writer.py`	13 lines (ranges: 156-158, 312, 314-315, 317-322, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_create_writer` 1ms 4

AI ASSESSMENT

Scenario: Test that a new `ReportWriter` instance is created with the provided configuration.

Why Needed: This test prevents a potential bug where the `ReportWriter` instance does not initialize correctly with the provided configuration.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	3 lines (ranges: 156-158)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_assembles_tests` 9ms 4

AI ASSESSMENT

Scenario: Test writes report with assembled tests.

Why Needed: This test prevents a regression where the report does not include all tests and only shows failed ones.

Key Assertions:

- The length of the report.tests list should be equal to 2.
- The value of report.summary.total should be equal to 2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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` 10ms 4

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class writes a report with a total coverage percentage.

Why Needed: This test prevents a regression where the reported coverage percentage is not accurate due to missing or incomplete data.

Key Assertions:

- The `coverage_total_percent` attribute of the `report.summary` object should be equal to the provided `coverage_percent` value.
- The `report.summary.coverage_total_percent` attribute should be a floating point number between 0 and 100.
- The `report.summary.coverage_total_percent` attribute should not exceed the maximum possible coverage percentage (100%).
- The `report.summary.coverage_total_percent` attribute should be greater than or equal to the minimum required coverage percentage (90%).
- The `report.summary.coverage_total_percent` attribute should be a non-negative number.
- The `report.summary.coverage_total_percent` attribute should not be zero.
- The `report.summary.coverage_total_percent` attribute should be an integer value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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` 9ms 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 by ensuring that the report accurately reflects source coverage.

Key Assertions:

- The length of `report.source_coverage` should be 1.
- The file path of the first element in `report.source_coverage` should match `file_path = "src/foo.py"`.
- All elements in `report.source_coverage` should have a valid `file_path`, `statements`, `missed`, `covered`, `coverage_percent`, and `covered_ranges` 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	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` 9ms 4

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_merges_coverage verifies that the report writer merges coverage into tests.

Why Needed: This test prevents regression where the coverage is not properly merged into the test results.

Key Assertions:

- The number of coverage entries in the first test should be 1.
- The file path of the first coverage entry should match 'src/foo.py'.
- Each coverage entry's file path should have a line range of '1-5' and a line count of 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	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` 10ms 5

AI ASSESSMENT

Scenario: Test that the ReportWriter falls back to direct write if atomic write fails.

Why Needed: To prevent a regression where the atomic write operation fails and the fallback to direct write is not executed.

Key Assertions:

- The report.json file 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` 11ms 5

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` creates a directory if it does not exist.

Why Needed: This test prevents a potential issue where the report writer fails to create an output directory when the input JSON file is missing.

Key Assertions:

- The output directory should be created at the specified path.
- The `ReportWriter` should raise an exception if the input JSON file does not exist.
- The test should fail with a meaningful error message indicating that the report writer failed to create the output directory.
- The test should verify that the output directory is correctly created and contains the expected files.
- The test should check that the `ReportWriter` raises an exception when the input JSON file does not exist.
- The test should verify that the error message indicates a failure to create the output directory.
- The test should check that the error message includes information about the missing input 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	84 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 229-231, 233, 235, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	123 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345,

	348-349, 352-354, 357, 360-364, 470-477, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_ensure_dir_failure` 1ms 4

AI ASSESSMENT

Scenario: Test verifies that the `test_ensure_dir_failure` test case ensures a directory creation failure is captured by the report writer.

Why Needed: This test prevents a regression where the report writer does not capture directory creation failures, potentially leading to silent errors or incorrect reporting of such issues.

Key Assertions:

- The `writer.warnings` list contains at least one warning with code 'W201' (indicating a permission denied error).
- The `writer.warnings` list is not empty.
- The `writer.warnings` list does not contain any warnings with code other than 'W201'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	12 lines (ranges: 156-158, 470-473, 480-484)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_git_info_failure` 1ms 3

AI ASSESSMENT

Scenario: Test 'test_git_info_failure' verifies that the test_report_writer.py module can handle git command failures gracefully.

Why Needed: This test prevents a potential bug where the test_report_writer.py module does not properly handle git command failures, potentially leading to unexpected behavior or errors.

Key Assertions:

- The 'get_git_info()' function should return None for both sha and dirty values when git is not found.
- The 'sha' variable should be None after calling 'get_git_info()'.
- The 'dirty' variable should also be None after calling 'get_git_info()'.
- An exception of type Exception should be raised when calling 'subprocess.check_output' with a side effect.
- The exception should have the message 'Git not found'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
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` 36ms 5

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class creates an HTML file with expected content.

Why Needed: This test prevents a regression where the report writer does not create an HTML file, potentially causing issues with report generation.

Key Assertions:

- The report.html file should exist at the specified path.
- The report.html file should contain the expected content as described in the assertions.
- All nodes in the report should be marked as either 'PASSED', 'FAILED', 'Skipped', 'XFailed', or 'XPassed'.
- Errors and warnings should not be present in the report.
- Test1 should be reported as passed, Test2 should be reported as failed with an error message.
- All test nodes should have a nodeid of either 'test1' or 'test2'.

COVERAGE

src/pytest_llm_report/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` 37ms 5

AI ASSESSMENT

Scenario: The test verifies that the xfail outcomes are included in the HTML summary.

Why Needed: This test prevents a regression where the xfail outcomes are not displayed in the HTML summary.

Key Assertions:

- The 'XFAILED' and 'XFailed' tags should be present in the HTML report.
- The 'XPASSED' and 'XPassed' tags should also be present in the HTML report.
- All xfail outcomes should be included in the HTML summary, regardless of their outcome status.

COVERAGE

src/pytest_llm_report/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` 10ms 5

AI ASSESSMENT

Scenario: Test verifies that a JSON file is created with the report.

Why Needed: This test prevents regression where the report writer does not create a JSON file.

Key Assertions:

- The 'report.json' file should be created in the specified path.
- At least one artifact should be tracked for the report.
- The length of the artifacts list should be greater than zero.
- The file 'report.json' exists at the specified path.
- The 'report.json' file contains the expected JSON 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	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` 39ms 5

AI ASSESSMENT

Scenario: Test verifies that the `write_pdf` method creates a PDF file when Playwright is available.

Why Needed: This test prevents regression where the `write_pdf` method does not create a PDF file even though Playwright is available.

Key Assertions:

- The `writer.write_report(tests)` call should successfully write the report to the specified PDF file.
- Any artifacts created by the report writer should have paths that match the path of the PDF file.
- The `exists()` method on the PDF file path should return True.
- The `any()` function should find any artifact with a path equal to the PDF file path.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/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` 10ms 4

AI ASSESSMENT

Scenario: Test verifies that a warning is raised when the 'report_pdf' configuration option is set to a file path without a Playwright installation.

Why Needed: This test prevents a potential issue where the report writer does not warn users about missing Playwright for PDF output, potentially leading to confusion or errors in their workflow.

Key Assertions:

- The 'report_pdf' configuration option is set to a file path without a Playwright installation.
- The 'code' attribute of each warning object matches WarningCode.W204_PDF_PLAYWRIGHT_MISSING.value.
- Any warnings raised by the report writer do not have an empty 'code' attribute.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
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 directory creation warning when writing reports with the `report_html` parameter.

Key Assertions:

- The test verifies that the `tmp_test_dir` exists after creating it.
- The test verifies that any warnings from the report writer are equal to 'W202'.
- The test verifies that the `tmp_test_dir` is deleted after writing the reports.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 18ms 5

AI ASSESSMENT

Scenario: Verify that the report_writer_metadata_skips test correctly skips metadata when reports are disabled.

Why Needed: This test prevents a regression where metadata is included in reports even when they are disabled.

Key Assertions:

- The 'start_time' key should be present in the metadata.
- The 'llm_model' key should not be present in the metadata if the report is disabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	36 lines (ranges: 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	67 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300)

PASSED tests/test_schemas.py::TestAnnotationSchema::test_from_dict_full 1ms 3

AI ASSESSMENT

Scenario: Test that `AnnotationSchema.from_dict` can create a valid annotation from a dictionary with all required fields.

Why Needed: Prevents regression in case of missing or invalid input data, ensuring the application behaves correctly when receiving unstructured input.

Key Assertions:

- assert schema.scenario == 'Verify login'
- assert schema.why_needed == 'Catch auth bugs'
- assert schema.key_assertions == ['assert 200', 'assert token']
- assert schema.confidence == 0.95

COVERAGE

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

PASSED tests/test_schemas.py::TestAnnotationSchema::test_to_dict_full 1ms 3

AI ASSESSMENT

Scenario: test_to_dict_full verifies that the full annotation to dictionary conversion is successful.

Why Needed: This test prevents regression by ensuring that the full annotation to dictionary conversion correctly captures all necessary fields, including scenario and why needed information.

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` 88ms 7

AI ASSESSMENT

Scenario: The HTML report is generated successfully.

Why Needed: This test prevents a regression where the report does not exist or has incorrect content.

Key Assertions:

- The file path of the report should be created.
- The report should contain the expected string '
- The report should contain the expected string 'test_simple'.

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` 125ms 7

AI ASSESSMENT

Scenario: test_html_summary_counts_all_statuses verifies that the HTML summary counts include all statuses.

Why Needed: This test prevents regression where the report does not display the total number of tests, failed tests, skipped tests, and errors.

Key Assertions:

- The 'Total Tests' card should contain the correct count.
- The 'Passed' card should contain the correct count.
- The 'Failed' card should contain the correct count.
- The 'Skipped' card should contain the correct count.
- The 'XFailed' and 'XPassed' cards should contain the correct counts.
- The 'Errors' card should contain the correct count.
- All labels in the list passed should match the expected count.

COVERAGE

src/pytest_llm_report/collector.py	65 lines (ranges: 78-79, 90, 93-94, 96, 99-104, 106-107, 109-112, 114-119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212-214, 216, 227-228, 230-236, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213,

	238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	111 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-328, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_json_report_created` 74ms 7

AI ASSESSMENT

Scenario: The JSON report is created successfully.

Why Needed: This test prevents a regression where the report generation fails due to an incorrect or missing schema version.

Key Assertions:

- The 'schema_version' key in the report data should be set to '1.0'.
- The 'summary' dictionary should contain the correct keys ('total', 'passed', and 'failed') with the expected values.
- The number of 'passed' and 'failed' counts in the summary should match the total count in the report.

COVERAGE

src/pytest_llm_report/collector.py	51 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-118, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 227-228, 230-236, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151,

	153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/report_writer.py	107 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_llm_annotations_in_report` 82ms 13

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report when a provider is enabled.

Why Needed: Prevents regression by ensuring LLM annotations are present in reports.

Key Assertions:

- The 'scenario' key in the report contains 'Checks the happy path'.
- The 'why_needed' key in the report indicates that LLM annotations prevent regressions.
- The 'llm_annotation' key in the report is present and contains the expected data.
- The 'scenario' value matches the provided string 'Checks the happy path'.
- The 'key_assertions' list includes the expected assertions for the 'llm_annotation' key.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	39 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198, 203)
src/pytest_llm_report/llm/base.py	39 lines (ranges: 52-53, 72, 75, 80, 107, 110-111, 128, 136, 147, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218, 245, 247, 249, 252, 257-258, 260)
src/pytest_llm_report/llm/litellm_provider.py	23 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 66-67, 69-70, 94-95, 97)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)

src/pytest_llm_report/models.py	94 lines (ranges: 104-107, 109-111, 113, 115, 161-165, 167, 169, 171, 173, 176, 178-180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407-419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	47 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	186 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-340, 343, 345, 348-352, 355, 357-362, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/prompts.py	29 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-109, 111-112, 116)
src/pytest_llm_report/report_writer.py	105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-296, 298-299, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_llm_error_is_reported` 6.09s 12

AI ASSESSMENT

Scenario: Test that LLM errors are surfaced in HTML output.

Why Needed: This test prevents regression where LLM errors are not reported correctly.

Key Assertions:

- The 'LLM error' keyword is present in the report content.
- The 'boom' string is present in the report content.
- The 'LLM error' and 'boom' keywords are both found in the report content.

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 59ms 7

AI ASSESSMENT

Scenario: Test the LLM opt-out marker.

Why Needed: Prevents a regression where the LLM opt-out marker is not recorded correctly.

Key Assertions:

- The test verifies that the `llm_opt_out` marker is applied to the `test_opt_out` function.
- The test verifies that the `llm_opt_out` marker is set to `True` for the `test_opt_out` function.
- The test verifies that only one test is recorded in the report.
- The test verifies that the `llm_opt_out` marker is correctly applied to the `test_opt_out` function.
- The test verifies that the `llm_opt_out` marker is not set for other functions.
- The test verifies that the `llm_opt_out` marker is recorded in the report correctly.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181-182, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180-182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134,

	136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/report_writer.py	105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED tests/test_smoke_pytester.py::TestMarkers::test_requirement_marker 58ms 7

AI ASSESSMENT

Scenario: Test the requirement marker to ensure it records the correct requirements.

Why Needed: This test prevents a potential bug where the requirement marker is not recorded correctly, potentially leading to incorrect reporting of tests that require specific dependencies.

Key Assertions:

- The `pytest.mark.requirement` decorator should be applied to the test function with the required requirements.
- The `test_with_req()` function should have the correct requirements specified using the `@pytest.mark.requirement()` decorator.
- The report generated by pytest should contain the correct requirements for each test.
- The `reqs` list in the `tests` dictionary should contain the expected requirements.
- The 'REQ-001' and 'REQ-002' strings should be present in the `reqs` list.
- The `test_with_req()` function should have a valid requirement string for each test.
- The report path should contain the correct file name and path to the report JSON.

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` 65ms 7

AI ASSESSMENT

Scenario: The 'test_multiple_xfail_outcomes' test verifies that multiple xfailed tests are recorded in the report.

Why Needed: This test prevents regression by ensuring that all xfailed tests are properly reported and counted in the report.

Key Assertions:

- The number of xfailed tests is exactly 2 as per the expected output.
- Each xfailed test has a corresponding outcome ('xfailed') in the 'outcomes' list.
- All xfailed tests are included in the 'tests' list with their respective outcomes.
- The 'summary' dictionary contains an 'xfailed' key with a value of 2, indicating that there were exactly 2 xfailed tests.
- Each test is marked as xfail using the '@pytest.mark.xfail' decorator.
- The 'makepyfile' function creates two test functions ('test_xfail_one' and 'test_xfail_two') with the '@pytest.mark.xfail' decorator.

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 66ms 7

AI ASSESSMENT

Scenario: Verifies that the test skip outcome is correctly recorded in the report.

Why Needed: This test prevents a false positive 'all tests passed' message when a test is skipped.

Key Assertions:

- The number of skipped tests should be 1 according to the report.
- The report path contains the expected file name for skipped tests (report.json).
- The data dictionary in the report should contain the correct key 'skipped' with value 1.
- The assertion checks if the 'summary' key exists and has a value of 1.
- The assertion checks if the 'skipped' count is equal to 1.
- The assertion checks if the file name contains the expected string (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 62ms 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 an Xfailed test does not get recorded in the report.

Key Assertions:

- The test 'test_xfail' should be marked as Xfailed by the pytester.
- The test 'test_xfail' should have an outcome of Xfailed in the report.
- The number of Xfailed tests should match the expected value (1) in the report.

COVERAGE

src/pytest_llm_report/collector.py	47 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-116, 119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212, 216, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190,

	192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/report_writer.py	108 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321, 323-324, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestParametrization::test_parametrized_tests` 64ms 7

AI ASSESSMENT

Scenario: The test verifies that parameterized tests are recorded separately and their results are correctly reported in the report.json file.

Why Needed: This test prevents a regression where parameterized tests might not be recorded or reported correctly, leading to incorrect reporting of test results.

Key Assertions:

- Parameterized tests are recorded separately
- The total count of passed tests is correct (3)
- The number of passed tests matches the expected value (3)
- The report.json file contains the correct summary data

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 51ms 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	45 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	118 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 387-388, 391, 395-397)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_markers_registered` 46ms 3

AI ASSESSMENT

Scenario: Verify that LLM markers are registered and correctly displayed in the pytest output.

Why Needed: This test prevents a potential issue where LLM markers are not properly registered, potentially causing unexpected behavior or errors in downstream tests.

Key Assertions:

- The 'llm_opt_out*' marker should be present in the pytest output.
- The 'llm_context*' marker should be present in the pytest output.
- The 'requirement*' marker should be present in the pytest output.
- The 'llm_opt_out*' marker should match the expected pattern in the pytest output.
- The 'llm_context*' marker should match the expected pattern in the pytest output.
- The 'requirement*' marker should match the expected pattern in the pytest output.
- All markers should be present in the pytest output, without any missing or incorrect matches.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	45 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	118 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 387-388, 391, 395-397)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_plugin_registered` 53ms 3

AI ASSESSMENT

Scenario: The plugin is successfully registered and its help message is displayed.

Why Needed: This test prevents a potential issue where the plugin registration fails or is not properly reported by pytest11.

Key Assertions:

- The `--help` option should display the help message of the plugin.
- The output should contain the line '*--llm-report*'.
- The plugin's name and description should be displayed in the help message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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` 85ms 7

AI ASSESSMENT

Scenario: Test 'Special characters in nodeid' verifies that the test does not crash when special characters are present in node IDs.

Why Needed: This test prevents a potential bug where special characters in node IDs could cause the test to crash or produce invalid results.

Key Assertions:

- The report path should exist and contain '' as part of its contents.
- The HTML file should be valid and not contain any syntax errors.
- Special characters should not be present in the node ID, causing the test to fail.

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: This test prevents regression in cases where the input duration is less than or equal to one minute, as it would incorrectly format it as '1m 0.0s'.

Key Assertions:

- The result should be '1m 0.0s' when the input duration is exactly 60 seconds.
- The unit prefix 'm' should not be included in the output if the input duration is less than or equal to one minute.
- The total number of seconds should match the input duration.
- Any non-numeric characters (e.g., 's') should be ignored when formatting the result.
- The function should raise a `ValueError` for invalid input durations (e.g., negative numbers, zero).
- The function should handle cases where the input duration is exactly one minute without any additional units.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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: Verifies that the function `format_duration` correctly formats sub-millisecond durations as microseconds.

Why Needed: This test prevents a potential bug where the function does not format durations to include microsecond units.

Key Assertions:

- The result of calling `format_duration(0.0005)` should contain the string 'µs'.
- The formatted output should be equal to '500µs'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	2 lines (ranges: 39-40)

PASSED `tests/test_time.py::TestFormatDuration::test_milliseconds_format` 1ms 3

AI ASSESSMENT

Scenario: Test that the `format_duration` function correctly formats sub-second durations as milliseconds.

Why Needed: This test prevents a potential bug where the function does not format the duration to the correct number of decimal places when it is less than one second.

Key Assertions:

- The result should contain 'ms' in its string representation.
- The value of `result` should be equal to '500.0ms'.
- The function should handle durations less than one second correctly and format them as 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_minutes_format` 1ms 3

AI ASSESSMENT

Scenario: Test the 'minutes' key assertion for durations over a minute.

Why Needed: This test prevents regression when formatting durations to show minutes.

Key Assertions:

- The 'result' string should contain the word 'm'.
- The 'result' string should be in the format '1m X.Xs'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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: Verifies the correct formatting of multiple minutes in a duration.

Why Needed: Prevents incorrect formatting of durations with multiple minutes, which can lead to misleading output.

Key Assertions:

- The result should be '3m 5.0s' (three minutes and five seconds).
- The minute part should be displayed as '3'.
- The second part of the minute should be displayed as '5.0'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

PASSED `tests/test_time.py::TestFormatDuration::test_one_second` 1ms 3

AI ASSESSMENT

Scenario: Verifies that the function formats a single second into '1.00s' as expected.

Why Needed: Prevents regression in formatting time intervals to seconds.

Key Assertions:

- The function should return '1.00s' when given an argument of exactly 1.0.
- The function should handle cases where the input is not a number (e.g., strings, None).
- The function should correctly format negative time intervals (e.g., '-1.0s').
- The function should return '1.00' when given an argument of exactly 1.0 and no units.
- The function should handle cases where the input is a very large or small number (e.g., floats, 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/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED tests/test_time.py::TestFormatDuration::test_seconds_format 1ms 3

AI ASSESSMENT

Scenario: Verifies that the function correctly formats seconds under a minute.

Why Needed: This test prevents a potential bug where the function does not format seconds correctly when they are less than one minute.

Key Assertions:

- The result of `format_duration(5.5)` should contain the string 's' to indicate that it's in seconds.
- The result of `format_duration(5.5)` should be equal to '5.50s' to verify its correctness.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED tests/test_time.py::TestFormatDuration::test_small_milliseconds 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 1 millisecond.

Why Needed: This test prevents a regression where the function incorrectly returns '1ms' for durations less than 1 millisecond.

Key Assertions:

- The function should return '1.0ms' for a duration of 1 millisecond.
- The function should not return '1ms' or any other value for a duration less than 1 millisecond.
- The function should handle durations in the range [1, 1000000] 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	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 very small duration.

Why Needed: This test prevents a potential bug where the function incorrectly formats durations as milliseconds instead of microseconds when they are extremely small.

Key Assertions:

- The function should correctly format the given duration as microseconds.
- The function should return '1µs' for a duration of 1 microsecond.
- The function should handle very small durations (less than 0.001 seconds) without truncation or incorrect formatting.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	2 lines (ranges: 39-40)

PASSED tests/test_time.py::TestIsoFormat::test_formats_datetime_with_utc 1ms 3

AI ASSESSMENT

Scenario: Test the `iso_format` function with a datetime object representing UTC time.

Why Needed: This test prevents potential issues where a datetime object is not correctly formatted as UTC.

Key Assertions:

- The output of the `iso_format` function should be in the format 'YYYY-MM-DDTHH:MM:SS+HH:MM:SS', which represents UTC time.
- The `tzinfo` parameter should be set to `UTC` for the datetime object.
- Any errors or exceptions thrown by the `iso_format` function should be properly handled and reported.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

PASSED tests/test_time.py::TestIsoFormat::test_formats_naive_datetime 1ms 3

AI ASSESSMENT

Scenario: Test the naive datetime format by verifying it matches the expected output without timezone.

Why Needed: Prevents a potential bug where the naive datetime format is not correctly handled, potentially leading to incorrect date representation.

Key Assertions:

- The function `iso_format(dt)` returns the correct ISO formatted string for the given datetime object.
- The resulting string does not include any timezone information.
- The resulting string is in the correct format according to the ISO 8601 standard.
- The function handles edge cases where the input datetime is invalid or outside the expected range.
- The function correctly formats the naive datetime without considering daylight saving time (DST) rules.
- The function does not introduce any timezone-related bugs or inconsistencies.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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 dates with microseconds.

Key Assertions:

- The 'result' variable should contain the string '123456'.
- The 'result' variable should be equal to '123456'.
- The 'result' variable should include the substring '123456' in its value.
- The `iso_format` function is correctly handling microseconds by including them in the output.
- The `iso_format` function is not returning an empty string or None when given a datetime object with microseconds.
- The `iso_format` function is correctly formatting the date and time to include microseconds.
- The `iso_format` function is preserving the original timezone of the input datetime object.

COVERAGE

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

Why Needed: This test prevents a potential issue where the test may fail if the system's default timezone is not set to UTC.

Key Assertions:

- result.tzinfo is not None
- result.tzinfo == UTC
- result.tzinfo is a valid timezone object (e.g., '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.

Why Needed: Prevents incorrect or outdated times from being reported as current.

Key Assertions:

- The returned time is within the UTC time range (before <= result <= after).
- The difference between before and after is less than or equal to the specified tolerance.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)

PASSED tests/test_time.py::TestUtcNow::test_returns_datetime 1ms 3

AI ASSESSMENT

Scenario: The `utc_now()` function should return a valid datetime object.

Why Needed: This test prevents a potential bug where the function returns an incorrect or missing datetime value.

Key Assertions:

- assert isinstance(result, datetime)
- assert result is not None
- assert isinstance(result, datetime) and result.year == 1970
- assert isinstance(result, datetime) and result.month in [1, 2, 3, 4, 5]
- assert isinstance(result, datetime) and result.day in [1, 2, 3, 4, 5]
- assert isinstance(result, datetime) and result.hour == 0
- assert isinstance(result, datetime) and result.minute == 0

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)

Source Coverage

FILE	STMTS	MISS	COVER	%	COVERED LINES	MISSED LINES
src/pytest_llm_report/_git_info.py	2	0	2	100.0%	2-3	-
src/pytest_llm_report/aggregation.py	116	5	111	95.69%	13, 15-19, 21, 35, 38, 44, 46, 52-53, 55-57, 59, 61-64, 69, 73-74, 77-80, 84, 87-89, 93, 103, 109-111, 113-117, 119-120, 125, 127-128, 130-131, 134-135, 141-144, 146, 148, 162, 164, 168, 170, 172, 182, 184-188, 190-191, 194,	66, 90-91, 192, 203

					196, 205, 217, 219-233, 235, 237, 245-246, 248-249, 251, 253-255, 259, 262-263, 265-266, 269-271, 273, 275-276, 280	
src/pytest_llm_report/cache.py	47	3	44	93.62%	13, 15-19, 21, 27, 33, 39-41, 43, 53, 55-56, 58, 60-62, 68-69, 78, 86, 88, 90, 92, 94, 97, 103, 107, 118-119, 121, 123, 129, 132-136, 141, 144, 153	64-65, 130
src/pytest_llm_report/collector.py	111	2	109	98.2%	19, 21-22, 24, 26-27, 33-34, 45-50, 52, 58, 60-62, 69, 78-79, 81, 90, 93-94, 96, 99-104, 106-107, 109-112, 114-119, 121-122, 124, 127-128, 130, 132-133, 135-137, 140, 143, 155, 163-164, 167-169, 171, 173, 181-182, 185-189, 191, 198-200, 202, 209-210, 212-214, 216, 218, 227-228, 230-236, 238, 241, 250-252, 254, 261, 264-265, 268-269, 271, 277, 279, 285	141, 239

src/pytest_llm_report/coverage_map.py	135	10	125	92.59%	13, 15-17, 19-22, 30, 38, 44-45, 47, 58-60, 64, 72-73, 83, 86, 88-90, 92, 94-96, 98, 101-104, 106-108, 114, 116, 118, 121-122, 127, 131-135, 137-140, 144-146, 148, 150, 152-153, 156, 160-162, 165, 167-168, 173, 176, 178-184, 187-189, 191, 196, 199-200, 202, 204, 216-217, 220, 224-225, 228-234, 236, 239, 241, 243-244, 246-248, 250, 252-254, 259-260, 263-264, 271, 273, 276-279, 281-283, 285, 299-300, 302, 308	62, 123, 125, 128, 157, 221, 249, 251, 257, 274
src/pytest_llm_report/errors.py	35	0	35	100.0%	8-9, 12, 25-28, 31-36, 39-42, 45-46, 49-51, 54-55, 64-66, 68, 70, 74-76, 80, 129, 139	-
src/pytest_llm_report/llm/__init__.py	3	0	3	100.0%	4-5, 7	-

src/pytest_llm_report/llm/annotator.py	110	0	110	100.0%	4, 6-10, 12-15, 21-22, 25-28, 31, 45-46, 48-50, 54, 56-57, 59, 61-62, 64, 66-68, 71-72, 74-82, 87, 97-98, 100, 102, 104-105, 115, 127, 129-132, 137-139, 142, 165-168, 170-171, 176, 178, 180-183, 185-190, 192-193, 198-201, 203, 206, 229-232, 234, 236-237, 239-240, 245-246, 248-253, 255-256, 261-264, 266	-
src/pytest_llm_report/llm/base.py	78	0	78	100.0%	13, 15-18, 26, 40, 46, 52-53, 55, 72, 75-76, 78, 80, 101, 107-108, 110-111, 122, 128, 130, 136, 138, 147, 149, 165, 167-173, 175, 177, 186-187, 190-192, 194-195, 198-200, 203-208, 212, 214, 220-221, 224-225, 228-230, 233, 245, 247, 249-250, 252-253, 255, 257-258, 260, 262-263, 265, 267	-

src/pytest_llm_report/llm/gemini.py	275	18	257	93.45%	7, 9-13, 15-16, 23-27, 30-34, 37-42, 44-46, 48-50, 52, 57-63, 65-70, 72-73, 75-78, 80-85, 87-88, 91-97, 99-103, 105, 107-114, 121-122, 125, 128, 134, 136-139, 141-142, 144, 160-161, 167-169, 171-172, 174, 176-184, 186-188, 190-191, 193, 196, 200-208, 210-211, 213-215, 217-223, 225-227, 233-234, 238-239, 242-243, 245-248, 252-253, 260, 266-267, 269, 273-277, 279-283, 286-287, 292-293, 300-301, 303, 315, 317-318, 322, 327, 330-332, 335-343, 345-346, 348, 352-355, 357, 360-366, 368-374, 380-382, 384-387, 389, 391-392, 396-402, 405, 408-410, 412-414, 416-421, 427-428, 430-434, 437-440, 442-443, 445-447	89, 104, 106, 115-117, 199, 230-231, 235-237, 244, 250, 256, 367, 441, 444
src/pytest_llm_report/llm/litellm_provider.py	32	1	31	96.88%	7, 9, 11-12, 18, 21, 37-38, 44, 46, 49, 51-52, 54-56, 66-67, 69-70, 73, 76, 78-79, 81-82, 84, 88, 94-95, 97	74
src/pytest_llm_report/llm/noop.py	13	0	13	100.0%	8, 10, 12-13, 20, 26, 32, 34, 50, 52, 58, 60, 66	-
src/pytest_llm_report/llm/ollama.py	43	1	42	97.67%	7, 9, 11-12, 18, 24, 40-41, 47, 50, 52, 54-55, 57-60, 62-63, 66-67, 71-72, 74-75, 77, 81, 87-88, 90-92, 96, 102,	69

					104, 114, 116-117, 127, 132, 134-135	
src/pytest_llm_report/llm/schemas.py	36	1	35	97.22%	8, 10-12, 16, 22, 38, 42-44, 46-47, 50-53, 55, 58-59, 62-65, 67-68, 77, 84, 90, 94-98, 102, 130	39
src/pytest_llm_report/models.py	240	10	230	95.83%	17-18, 21, 24-25, 34-36, 38, 40, 47-48, 61-67, 69, 71, 82-83, 95-100, 102, 104, 109-115, 118-119, 141-157, 159, 161, 167-171, 173-182, 184, 186, 188-190, 193-194, 202-203, 205, 207, 213-214, 223-225, 227, 229, 233-235, 238-239, 248-250, 252, 254, 261-262, 271-273, 275, 277, 281-283, 286-287, 324-353, 355-360, 362, 364, 382-405, 407-419, 422-423, 437-445, 447, 449, 459, 461, 464-465, 482-492, 494, 500, 502, 508-512, 514, 516, 518, 520, 522	172, 183, 185, 187, 460, 513, 515, 517, 519, 521
src/pytest_llm_report/options.py	117	45	72	61.54%	106, 146, 175, 178-180, 185-187, 193-195, 201-203, 209-218, 220, 224, 233, 248, 251-267, 270-283, 286-295, 298, 300	13-15, 21-22, 90-94, 97-99, 102-105, 122-123, 126-132, 135-137, 140-142, 145, 156-160, 163-164, 167, 169, 222, 227, 236

src/pytest_llm_report/plugin.py	156	25	131	83.97%	40, 43, 49, 55, 61, 67, 73, 80, 89, 95, 101, 107, 113, 121, 126, 131, 136, 142, 147, 153, 169, 173, 177, 183-184, 187-188, 190, 192, 195-197, 203-204, 212-213, 238-239, 242-243, 246, 249-250, 252-253, 256-257, 259, 261-265, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-305, 307, 309-314, 317-318, 322-323, 331-332, 337-340, 343, 345, 348-353, 355, 357, 365-366, 387-388, 391-392, 395-397, 408-409, 412, 415-416, 419-421, 431-432, 435-437, 448-449, 452, 455, 457-458	13, 15-17, 19-20, 22, 28-31, 34, 160, 216, 319, 327-328, 333-334, 379-380, 400, 424, 440-441
src/pytest_llm_report/prompts.py	75	5	70	93.33%	13, 15-17, 24, 27, 33, 35, 49, 52, 55, 58-61, 63, 65, 67, 78-79, 82-84, 86-87, 92, 94-95, 98-101, 103-112, 116, 118, 132-133, 135-138, 140-141, 144-145, 148, 151-152, 154-156, 158-159, 163, 165, 180, 182, 191-194	80, 114, 142, 146, 149
src/pytest_llm_report/render.py	50	0	50	100.0%	13, 15-16, 18, 24, 30-31, 34, 40, 42, 50-51, 53, 56, 65-67, 70, 79, 87, 90, 99, 101-102, 107, 110, 121-124, 126-129, 131-134,	-

					141-143, 145, 158-163, 177, 196	
src/pytest_llm_report/report_writer.py	167	10	157	94.01%	13, 15-25, 27-29, 46, 55, 58, 67-68, 76, 83-84, 89, 98-100, 102, 105-108, 110, 116, 127-128, 130, 142, 150, 156-158, 160, 186-189, 192, 197-199, 202-203, 211, 222-223, 226-227, 230-231, 233, 235, 254, 256-259, 262-264, 266, 268, 303, 312, 314-315, 317-328, 330, 332, 340, 343-345, 348-349, 352-354, 357, 360, 368, 376, 378-379, 382, 385, 388, 391, 399, 401-402, 408, 410, 412, 414-423, 434-435, 437-439, 447-448, 453, 455, 458, 461-462, 464, 470-474, 480-481, 488, 495, 497, 499-501, 503, 506-507, 509, 515-516	113, 135-137, 424-425, 432, 449-451
src/pytest_llm_report/util/fs.py	34	3	31	91.18%	11, 13-14, 17, 30, 33, 36, 39, 42, 45, 55-56, 58-60, 63-64, 70, 79, 82, 100, 103, 111-113, 116-117, 119-121, 123	40, 65, 67
src/pytest_llm_report/util/hashing.py	36	0	36	100.0%	12, 14-17, 23, 32, 35, 44-48, 51, 61, 64, 73-74, 76-78, 80-81, 86, 96, 103-104, 107, 113-114, 116-121	-

src/pytest_llm_report/util/ranges.py	33	0	33	100.0%	12, 15, 29-30, 33, 35-37, 39-40, 42, 45-47, 50, 52, 55, 65-67, 70, 81-82, 84-91, 93, 95	-
src/pytest_llm_report/util/time.py	16	0	16	100.0%	4, 6, 9, 15, 18, 27, 30, 39-44, 46-48	-