Test Report

Run ID: 20947644922-py3.12 • Generated: 2026-01-13 06:57:28 • Duration: 34.37s

Plugin: v0.1.0 (2f498263985a34902252c53c11fb820445bd8f21) [dirty]

Repo: v0.1.0 (b9e3e95cf545147fb1baae5e547e056f45911b0a)

LLM: ollama / llama3.2:1b (minimal context, 383 annotated, 3 errors)

92.91%

Total Coverage

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

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

AI ASSESSMENT

Scenario: Test that the aggregate function correctly handles all policy when aggregating multiple reports.

Why Needed: This test prevents a regression where an aggregation of multiple reports with different policies would not retain all tests due to incorrect filtering.

Key Assertions:

- The aggregated report should have exactly two retained tests, regardless of the policy used.
- Each retained test should be from one of the original reports.
- The aggregate function should correctly filter out tests based on the aggregation policy.
- The aggregate function should not retain any tests that are explicitly excluded by the policy.
- The aggregate function should handle multiple policies correctly, allowing for different filtering rules to be applied.
- The test case should pass even if the policy is set to 'none' or 'partial'.
- The test case should fail if the policy is set to 'all' and one of the reports has a retained test that does not match the policy.

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: Verify that the aggregate function does not attempt to aggregate a non-existent directory.

Why Needed: Prevents a potential error where an empty or nonexistent directory is attempted to be aggregated.

Key Assertions:

- The `aggregate` method of the `aggregator` instance should return None when the provided directory does not exist.
- A `PathError` exception should not be raised if the directory does not exist.
- The aggregate function should not attempt to create or modify any files or directories within the non-existent directory.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the `aggregate_latest_policy` function correctly selects the latest report when there are multiple reports of the same test with different times.

Why Needed: This test prevents a regression where the `aggregate` function would incorrectly select the first report it encounters as the latest, instead of the most recent one.

Key Assertions:

- The `outcome` of the aggregated report is set to 'passed' if there are multiple reports of the same test with different times.
- The number of tests in the aggregated report is 1.
- The outcome of the first test in the aggregated report is 'passed'.
- The `run_meta.run_count` attribute is equal to 2.
- The `summary.passed` attribute is equal to 1.
- The `summary.failed` attribute is equal to 0.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The aggregator function should not throw an error when no directory configuration is provided.

Why Needed: This test prevents a potential bug where the aggregator function throws an exception when no directory configuration is specified.

Key Assertions:

- agg.aggregate() should return None
- agg.aggregate() should not raise an exception
- mock_config.aggregate_dir should be set to None

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that aggregate function returns None when no reports exist and no files are found.

Why Needed: Prevents regression where the aggregate function fails to return an empty list of reports when there are no files or reports available.

Key Assertions:

- The aggregate function should return None for this scenario.
- The aggregate function should not attempt to access any files or reports.
- There should be no error raised if the file system is empty.
- The aggregate function should handle a case where there are no reports available without raising an exception.
- The aggregate function should preserve the original behavior when called with a non-empty list of reports.
- The aggregate function should not attempt to access any files or reports in this scenario.
- There should be no assertion error raised if the file system is empty.
- The aggregate function should handle a case where there are no files found without raising an exception.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- coverage: Ensure coverage is correctly deserialized with the expected file paths and line ranges.
- LLM Annotation: Verify correct deserialization of LLM annotation with scenario, why-need, and key assertions.
- Re-serialization: Confirm that the aggregated report can be re-serialized with accurate coverage and LLM annotation.

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 correctly aggregates source coverage data from a temporary report.

Why Needed: This test prevents regression where the aggregation of source coverage data fails to produce accurate results due to incomplete or missing reports.

Key Assertions:

- The `source_coverage` attribute of the aggregated result is an instance of `SourceCoverageEntry`.
- The `file_path` attribute of the first `SourceCoverageEntry` in the aggregated result matches the expected file path.
- All statements in the source code are covered by at least 83.33% of the total coverage.
- At least 2 out of 10 statements were missed in the source code.
- The coverage percentage is between 1 and 5, inclusive for the first range and exclusive for the second range.
- The missing ranges are '6' and '12'.
- All covered statements are within the specified ranges.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the test fails due to an unconfigured source file.

Key Assertions:

- Verify that the `_load_coverage_from_source` method returns `None` when `llm_coverage_source` is `None`.
- Verify that the `_load_coverage_from_source` method raises a UserWarning with the message 'Coverage source not found' when `llm_coverage_source` is '/nonexistent/coverage'.
- Verify that the `_load_coverage_from_source` method returns `None` when `llm_coverage_source` does not exist.
- Verify that the mock coverage object is created and returned by the mock cov_cls before calling it.
- Verify that the mock mapper object is created and returned by the mock mapper_cls before calling it.
- Verify that the `map_source_coverage` method of the mock mapper returns a list with one entry.
- Verify that the `load` method of the mock cov_cls is called once.
- Verify that the `map_source_coverage` method of the mock mapper has been called with the mock cov object.

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 that the _recalculate_summary method preserves the total duration of tests and maintains correct coverage percentages.

Why Needed: This test prevents regression where the total duration is not preserved or the coverage percentage does not match the expected values after recalculating the summary.

Key Assertions:

- The total number of tests passed, failed, skipped, xfailed, xpassed, and error should be equal to the original counts.
- The total duration of all tests should remain unchanged.
- The coverage percentage should match the expected value for each test type (total, passed, failed, skipped, xfailed, xpassed, and error).

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: Skipping an invalid JSON aggregation report

Why Needed: Prevents a regression where a test fails due to an unexpected error when aggregating reports with non-JSON content.

Key Assertions:

- The `aggregate` function should skip the 'invalid.json' file because it does not contain valid JSON data.
- The `aggregate` function should only count the valid report ('valid.json') in its run meta.
- When an invalid JSON file is encountered, a UserWarning is raised with a message indicating that the aggregation skipped the file.
- The test verifies that the `aggregate` function correctly handles missing fields in the aggregated report.
- The test ensures that only the valid report is included in the run meta of the aggregated result.

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 coverage totals.

Why Needed: This test prevents regression in the aggregator's behavior when handling different coverage scenarios, ensuring accurate summary calculations.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: This test verifies that cached tests are skipped by the annotator.

Why Needed: To prevent regression in the annotator's caching behavior and ensure it only skips cached tests.

Key Assertions:

- The `mock_provider` is not called with any arguments.
- The `mock_cache` is not called with any arguments.
- The `mock_assembler` is not called with any arguments.
- The `test_cached_tests_are_skipped` method does not call the `cached_test` function.
- The `cached_test` function is not called by the `mock_provider` or `mock_cache`.
- The `mock_assembler` does not call the `annotator` instance before calling its methods.
- The `mock_provider` and `mock_cache` do not have any side effects that would cause them to be called with arguments.
- The `test_cached_tests_are_skipped` method is not called by other test functions in the same scope.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The annotator function is called concurrently without proper synchronization.

Why Needed: This test prevents a potential concurrency issue where multiple annotations are performed simultaneously, potentially leading to incorrect results or errors.

Key Assertions:

- mock_provider.call_args.assert_called_once_with(self, 'annotation', mock_assembler)
- mock_cache.call_args.assert_called_once_with(self, 'annotation', mock_assembler)
- mock_assembler.call_args.assert_called_once_with(self, 'annotation')

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	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 that concurrent annotation handles failures by verifying the annotator's behavior.

Why Needed: This test prevents a potential regression where the annotator fails to handle failures in a concurrent environment.

Key Assertions:

- The annotator should not fail when encountering an error while processing annotations concurrently.
- The annotator should log the error and continue processing other annotations without interruption.
- The annotator's cache should be updated correctly even if it encounters errors during annotation.
- The annotator's progress bar should update correctly even if it encounters errors during annotation.
- The annotator's output should not be affected by concurrent failures.
- The annotator's error messages should contain relevant information about the failure.
- The annotator's logging behavior should be consistent across different environments.

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: Verify that the progress reporting is correctly implemented in the annotate function.

Why Needed: This test prevents regressions where the progress reporting might not be accurately tracked or reported.

Key Assertions:

- The `progress` attribute of the annotated object should increase as the annotation progresses.
- The `total_progress` attribute of the annotator should correctly track the total number of annotations.
- The `annotation_id` attribute of each annotation should increment in a predictable manner.
- The `annotated_object` attribute of each annotation should be updated correctly after progress reporting.
- The `progress` and `total_progress` attributes of the annotator should be reset to 0 when no new annotations are added.
- The `annotation_id` attribute of the last annotated object should match its original value before the test started.
- The `annotated_object` attribute of the last annotated object should contain a valid annotation data.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verifies the sequential annotation functionality of the annotator.

Why Needed: Prevents regression in sequential annotation by ensuring that the annotator properly handles multiple annotations in a row.

Key Assertions:

- The `mock_provider` is called with a valid `annotator` instance.
- The `mock_cache` is not called during the execution of the test.
- No exceptions are raised when calling `mock_assembler` on the `annotator` instance.
- The `annotator` instance is properly updated after each annotation.
- No unexpected behavior occurs when annotating multiple items in a row.
- The `mock_provider` and `mock_cache` instances are not reused across tests.
- The `mock_assembler` instance is only called once during the execution 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/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the annotator skips tests when LLM is disabled.

Why Needed: This test prevents a regression where LLM is disabled and annotators are still executed.

Key Assertions:

- The function `annotate_tests` should not be called with an empty list of tests.
- The function `annotate_tests` should not call the `LLM` provider.
- The `LLM` provider should be set to 'none' before calling `annotate_tests`.
- If LLM is disabled, the annotator should do nothing and return without making any changes to the test results.
- The function `annotate_tests` should not modify the test results or report any changes.
- The function `annotate_tests` should only be called with a valid 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/llm/annotator.py	2 lines (ranges: 45-46)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The annotator skips tests when the provider is unavailable.

Why Needed: To prevent skipping of critical tests due to a provider's unavailability.

Key Assertions:

- Mocked Provider Mock
- Provider Unavailable Error
- Test Skipping Due to Provider Unavailability
- Annotator Behavior When Provider Unavailable
- Error Handling for Provider Unavailability
- Test Case Prioritization Based on Provider Availability

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: Tests annotator concurrent with progress and errors

Why Needed: Prevents a potential bug where the annotator fails to report progress or first error when annotating concurrently.

Key Assertions:

- The function should correctly report progress and first error in concurrent mode.
- The function should append 'Processing 2 test(s)' to the progress messages list.
- The function should include 'LLM annotation' in the progress messages list.
- The function should not fail to report any errors when annotating concurrently.
- The function should correctly handle scenarios where multiple tasks are annotated concurrently.
- The function should update the cache with the correct number of annotations.
- The function should return 2 annotations as expected when there are 2 concurrent tasks.
- The function should not raise an exception when annotating concurrently.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	28 lines (ranges: 229-232, 234, 236-237, 239-242, 245-246, 248-253, 255-258, 261-264, 266)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- The time.sleep() function was called.
- The monotonic() function returned a value greater than 1.0s.
- The time.sleep() function was called again.
- The monotonic() function returned a value less than or equal to 1.0s.
- The time.sleep() function did not call itself.
- The monotonic() function returned the correct interval (1.0s) after the first sleep call.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Should report progress for cached tests.

Why Needed: Prevents regression where the annotator does not report progress for cached tests.

Key Assertions:

- The `progress_msgs` list should contain any messages that indicate a cache hit (e.g. `(cache): test_cached`).
- The `progress_msgs` list should not be empty after running the test.
- Any message in the `progress_msgs` list should start with '(cache): ' to identify it as related to caching.
- The `progress_msgs` list should contain messages for all cached tests (not just one).

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the annotator fails to annotate tests when the provider is not available.

Why Needed: This test prevents a regression where the annotator would incorrectly report tests as successful even though they are unavailable.

Key Assertions:

- mocks.get_provider().is_available() returns False
- annotate_tests(tests, config) will print 'not available. Skipping annotations'
- tests[0].outcome is not 'passed' after the annotation fails
- mock_provider.is_available() was called with a return value of False
- mock_provider.return_value is set to mock_provider

COVERAGE

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

PASSED tests/test_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 will fail when a malformed JSON is passed.

Why Needed: This test prevents a potential bug where the function `test_base_parse_response_malformed_json_after_extract` fails due to an invalid JSON in the response, causing it to raise a `JSONDecodeError`.

Key Assertions:

- The function `_parse_response(response)` will return `None` when the input is malformed JSON.
- The error message returned by `annotation.error` should be 'Failed to parse LLM response as JSON'.
- The function `provider._parse_response(response)` will raise a `JSONDecodeError` exception 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	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 that the `test_base_parse_response_non_string_fields` function handles non-string fields in the response data correctly.

Why Needed: This test prevents a potential bug where the function incorrectly assumes all fields are strings when they may not be.

Key Assertions:

- The function should return an annotation with the correct scenario value.
- The function should return an annotation with the correct why_needed list value.
- The function should return an annotation with the correct key_assertion list value.
- The `scenario` attribute of the annotation should be set to '123'.
- The `why_needed` attribute of the annotation should be set to ['list'].
- The `key_assertions` attribute of the annotation should contain the string '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: Verifies that the `get_gemini_provider` function returns a `GeminiProvider` instance.

Why Needed: Prevents a potential bug where the test fails if the `gemini` provider is not properly configured or if there's an issue with the Gemini API.

Key Assertions:

- The returned `provider` object should be an instance of `GeminiProvider`.
- The `provider` attribute of the returned `provider` object should contain a valid `GeminiProvider` instance.
- The `provider` attribute of the returned `provider` object should have a value that is not `None` or `undefined`.
- The `provider` attribute of the returned `provider` object should be an instance of `GeminiProvider` with the correct class name.
- The `provider` attribute of the returned `provider` object should contain the correct attributes (e.g., `name`, `api_key`, etc.).

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents the introduction of an Unknown LLM provider error in future code changes.

Key Assertions:

- The function `get_provider` raises a ValueError with the message 'Unknown LLM provider: invalid'.
- The exception type is `ValueError`.
- The exception message contains the string 'Unknown LLM provider: invalid'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verifies that the `get_litellm_provider` function returns a LitELLMProvider instance.

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

Key Assertions:

- The function `get_provider(config)` returns an instance of `LiteLLMProvider`.
- The `provider` attribute of the returned `LiteLLMProvider` instance is set to `'litellm'`.
- An exception is not raised if the provider is not set to 'litellm'.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents regression in case a new provider is added without updating the test.

Key Assertions:

- The returned value should be an instance of `NoopProvider`.
- The `provider` attribute of the returned value should not be `None`.
- The type of the returned value should match the expected type, which is `NoopProvider`.
- The `get_provider` function should return a valid provider instance when no provider is specified.
- A new provider should not break the test if it's added without updating the test.
- The test should pass even after adding a new provider to the system.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the `get_ollama_provider` function fails to return an instance of OllamaProvider due to incorrect configuration.

Key Assertions:

- The function `get_provider(config)` is called with the correct provider name 'ollama'.
- The returned value `provider` is an instance of `OllamaProvider`.
- The type of `provider` is correctly set to `OllamaProvider` using the `isinstance()` function.
- An error message or exception is not raised when calling `get_provider(config)` with the correct configuration.
- The provider name 'ollama' is properly formatted and does not contain any typos.
- The config object passed to `get_provider(config)` has the required attributes (provider) set correctly.
- A custom error message or exception is not raised when calling `get_provider(config)` with the correct configuration.
- The provider name 'ollama' is properly formatted and does not contain any leading or trailing whitespace.

COVERAGE

`src/pytest_llm_report/collector.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: Verifies that the LlmProvider defaults implementation provides a valid cache result.

Why Needed: The current implementation of LlmProvider does not provide a valid cache result, which can cause unexpected behavior in downstream applications.

Key Assertions:

- The `is_available()` method returns True for both cached and uncached configurations.
- The `checks` attribute is incremented correctly when the `_check_availability()` method is called.
- The `is_available()` method returns False for a cache configuration that does not have any available caches.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 107-108, 110-111)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The 'get_model_name' method of the ConcreteProvider class returns the name set in the test model configuration.

Why Needed: Without this default configuration, the LLM model may not be properly initialized or configured.

Key Assertions:

- the 'model' attribute is set to 'test-model'
- the 'name' attribute of the provider object is equal to 'test-model'
- the provider's get_model_name() method returns 'test-model'
- the configuration's model attribute has been successfully loaded
- the configuration's name attribute matches the expected default value
- the provider's name attribute does not match the expected default value

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	3 lines (ranges: 52-53, 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 the default rate limit for LLM providers is not properly set to None when no custom rate limits are provided.

Key Assertions:

- The `get_rate_limits()` method of the provider instance returns an empty list or None.
- The `rate_limit` attribute of the provider instance does not contain any valid numerical values.
- The `max_rate` and `min_rate` attributes of the provider instance are set to a default value (e.g., 1.0) that is not compatible with the provided rate limits.
- The `rate_limits` attribute of the provider instance contains invalid or unsupported types (e.g., strings, booleans).
- The `max_rate_limit` and `min_rate_limit` attributes of the provider instance are set to a default value that exceeds the maximum allowed rate limit for the specific LLM model.
- The `rate_limits` attribute is not updated when custom rate limits are provided in the configuration.
- The `get_rate_limits()` method raises an exception (e.g., ValueError) if no valid rate limits can be determined from the provider instance.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that `is_local()` returns False when default defaults are used.

Why Needed: Prevents regression in case of default defaults being used.

Key Assertions:

- The `provider` object is not local.
- The `provider` object has a non-local configuration.
- The `provider.is_local()` method returns False.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Testing the consistency of a cache with a single source.

Why Needed: To ensure that the cache maintains its expected behavior when using the same source function.

Key Assertions:

- The hash value of the source function should be the same every time it is called.
- The hash value of the source function should not change after multiple calls to the function.
- The cache should store the source function's return values with their corresponding hashes.
- The cache should maintain a consistent order of insertion for functions that have different hashes.
- The cache should handle function calls correctly, including caching the result of each call.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Testing the `hash_source` function with different sources.

Why Needed: Prevents a bug where two functions with the same source code but different names produce the same hash value.

Key Assertions:

- The `hash_source` function should return a different hash value for two different source strings.
- The `hash_source` function should not be able to deduce the source of a function from its name.
- The `hash_source` function should raise an error if given the same input multiple times.
- The `hash_source` function should preserve the original source code when hashing.
- The `hash_source` function should return different hash values for functions with the same name but different parameter lists.
- The `hash_source` function should not be able to deduce the source of a function from its docstring.
- The `hash_source` function should raise an error if given a function that does not have a `__name__` attribute.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify the length of the hash generated by `hash_source`.

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

Key Assertions:

- The hash should be at least 16 characters long.
- The hash should be exactly 16 characters long (e.g., 'aabbccdd').
- No leading zeros are allowed in the hash.
- No trailing zeros are allowed in the hash.
- The hash does not contain any repeated characters.
- All characters in the hash are unique.
- The hash is a valid SHA-256 hash (e.g., '1234567890abcdef').

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that clearing the cache removes all entries.

Why Needed: Prevents a bug where some cache entries are not properly cleared after test::a and test::b are removed.

Key Assertions:

- The number of cache entries should be exactly 2.
- test::a should not be present in the cache.
- test::b should not be present in the cache.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where error annotations are incorrectly cached.

Key Assertions:

- The annotation 'error' in the cache key 'test::foo' should be None.
- The annotation 'abc123' in the cache key 'test::foo' should not match any existing value.
- No error message should be stored in the cache for annotations with errors.

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: The test prevents a potential bug where the function does not handle missing cache entries correctly, potentially leading to incorrect results or errors.

Key Assertions:

- The function `cache.get()` should return `None` when called with an invalid key.
- The function `cache.get()` should raise a `KeyError` when called with an invalid key.
- The function `cache.get()` should not modify the cache when called with an existing key that is missing from it.

COVERAGE

src/pytest_llm_report/cache.py	9 lines (ranges: 39-41, 53, 55-56, 118-119, 121)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that annotations are stored and retrieved correctly from the cache.

Why Needed: Prevents bypass attacks by ensuring that annotations are cached before they can be used to bypass security measures.

Key Assertions:

- Check if the annotation is set with the correct key
- Check if the annotation's scenario matches the expected value
- Check if the confidence of the annotation is as expected
- Verify that the cache returns a non-None result for the given key
- Verify that the retrieved annotation has the correct confidence level

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 'nodeid' and 'message' attributes.

Why Needed: This test prevents a potential bug where a CollectionError is incorrectly structured, potentially leading to incorrect handling or reporting of errors in the collector.

Key Assertions:

- The 'nodeid' attribute of the error object should be equal to 'test_bad.py'.
- The 'message' attribute of the error object should be equal to '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 with an empty configuration.

Why Needed: This test prevents a potential regression where an empty collection is not immediately recognized as an error.

Key Assertions:

- The `get_collection_errors()` method returns an empty list when the input collection is empty.
- An empty collection is considered an error and should be handled accordingly in subsequent steps.
- A test asserting that an empty collection is returned is necessary to ensure the method behaves as expected in this scenario.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the default value of llm_context_override in TestCollectorMarkerExtraction.

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

Key Assertions:

- The llm_context_override attribute should be None for the given TestCaseResult.
- The llm_context_override attribute should not be set to any other value than None for the given TestCaseResult.
- If llm_context_override is set to a different value, it should be immediately overridden by the actual context.
- The default value of llm_context_override should be None for the given TestCaseResult.
- If the default value of llm_context_override is not None, it should be checked and verified in subsequent tests.
- The test should verify that the default value of llm_context_override is correctly set to None for the given TestCaseResult.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 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 correctly set to False for a test case.

Why Needed: This test prevents regression where the default value of llm_opt_out is incorrectly set to True.

Key Assertions:

- The llm_opt_out attribute should be set to False.
- The llm_opt_out attribute should not be set to True.
- The TestCaseResult object should have a llm_opt_out attribute with the correct value (False).
- The TestCaseResult object should not have a llm_opt_out attribute with an incorrect value (True).
- The llm_opt_out attribute should be correctly initialized in the TestCaseResult object.
- The TestCaseResult object should not have any other attributes that are not relevant to this test.
- The TestCaseResult object should pass all assertions without raising any exceptions.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- config.capture_failed_output should be set to False
- config.capture_enabled should be set to False
- output_capture_enabled should not be True

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/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: Verify that the default value of `capture_output_max_chars` is set to 4000.

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

Key Assertions:

- assert config.capture_output_max_chars == 4000
- assert config.capture_output_max_chars != 10000
- assert config.capture_output_max_chars != 5000
- assert config.capture_output_max_chars != 2000
- assert config.capture_output_max_chars != 3000
- assert config.capture_output_max_chars == 4000

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test 'xfail failures should be recorded as xfailed' verifies that failed xfail tests are correctly marked as xfailed in the report.

Why Needed: This test prevents regression by ensuring that failed xfail tests are properly recorded and marked as such, preventing incorrect reporting of failure status.

Key Assertions:

- The 'passed', 'skipped' and 'duration' fields of the SimpleNamespace object `report` should be set to False.
- The 'outcome' field of the SimpleNamespace object `result` should be set to 'xfailed'.
- The 'wasxfail' field of the SimpleNamespace object `report` should be set to 'expected failure'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test 'xfail passes should be recorded as xpassed' verifies that the test collector correctly records and reports xfail tests as passed.

Why Needed: This test prevents regression where an unexpected pass in a test is not properly reported as failed, potentially masking issues with test behavior or reporting.

Key Assertions:

- The `result.outcome` of the `test_unexpected_pass` node should be 'xpassed' after calling the `handle_runtest_logreport` method on the `TestCollector` instance.
- The `wasxfail` attribute of the `test_xfail.py::test_unexpected_pass` node should match the expected failure message.
- The `duration` and `longrepr` attributes of the `test_xfail.py::test_unexpected_pass` node should be set to 0.01 seconds and an empty string respectively after calling the `handle_runtest_logreport` method on the `TestCollector` instance.
- The `skipped` attribute of the `test_xfail.py::test_unexpected_pass` node should remain False after calling the `handle_runtest_logreport` method on the `TestCollector` instance.
- The `failed` attribute of the `test_xfail.py::test_unexpected_pass` node should be set to False after calling the `handle_runtest_logreport` method on the `TestCollector` instance.
- The `passed` attribute of the `test_xfail.py::test_unexpected_pass` node should remain True after calling the `handle_runtest_logreport` method on the `TestCollector` instance.
- After calling the `handle_runtest_logreport` method, the `config` object passed to the `TestCollector` constructor should not be modified.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the `TestCollector` class initializes correctly and returns an empty collection.

Why Needed: This test prevents a potential bug where the collector is initialized with incorrect or missing configuration settings, potentially leading to incorrect results.

Key Assertions:

- collector.results == {}
- collector.collection_errors == []
- collector.collected_count == 0

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the `get_results` method of TestCollector to ensure it returns sorted results by node ID.

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

Key Assertions:

- The list of node IDs returned by the `get_results` method is sorted in ascending order.
- The first element of the list is 'a_test.py::test_a'.
- The second element of the list is 'z_test.py::test_z'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `handle_collection_finish` method correctly tracks collected and deselected counts.

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

Key Assertions:

- The `collected_count` attribute of the collector should be equal to the number of items that were collected.
- The `deselected_count` attribute of the collector should be equal to the number of items that were deselected.
- The `collected_count` and `deselected_count` attributes should match the expected values after calling `handle_collection_finish` with a list of collected and deselected items.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test 'Should not capture if config disabled (integration via handle_runtest_logreport)' verifies that the test does not capture output when `capture_failed_output` is set to False.

Why Needed: This test prevents a bug where the test captures output even though `capture_failed_output` is set to False, which could lead to unexpected behavior or false positives in the test results.

Key Assertions:

- The 'nodeid' attribute of the report object should be empty.
- The 'outcome' attribute of the report object should be 'failed'.
- The 'when' attribute of the report object should be set to 'call'.
- The 'passed' attribute of the report object should be False.
- The 'failed' attribute of the report object should be True.
- The 'skipped' attribute of the report object should be False.
- The 'capstdout' attribute of the report object should be set to 'output'.
- The 'wasxfail' attribute of the report object should not exist.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the `test_capture_output_stderr` function does not capture stderr.

Key Assertions:

- The `captured_stderr` attribute of the `TestCaseResult` object is set to 'Some error'.
- The `report.capstderr` method is called with an argument equal to 'Some error'.
- The `collector._capture_output` function is called with a result and report object that have been populated with the expected values.

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 regression where the collector does not capture stdout when it should.

Key Assertions:

- The `captured_stdout` attribute of the `TestCaseResult` object is set to 'Some output'.
- The `report.capstdout` method returns 'Some output' as expected.
- The `report.capstderr` method does not interfere with the captured stdout.
- The collector correctly captures stdout even when it fails and reports an error.
- The `test_capture_output_stdout` function is able to distinguish between captured stdout and reported errors.
- The test result indicates that the collector successfully captured stdout.

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: The test verifies that the `TestCollector` truncates output exceeding the specified maximum characters.

Why Needed: This test prevents a potential issue where the collector fails to capture and display the entire output of a test, potentially leading to misleading results or errors.

Key Assertions:

- The captured stdout should be truncated at 10 characters.
- The captured stderr is empty.
- The `captured_stdout` attribute of the `TestCaseResult` object should contain only the first 10 characters of the output.
- The collector does not fail when capturing output that exceeds the maximum characters.
- The collector correctly truncates output and displays it in the report.
- The captured stderr is empty even if the output exceeds the maximum characters.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents regression by ensuring that the collector correctly extracts item markers from an item and returns them in the expected format.

Key Assertions:

- The `param_id` of the extracted marker is set to 'param1'.
- The value of `llm_opt_out` is set to True.
- The value of `llm_context_override` is set to 'complete'.
- A list of requirements is returned with values ['REQ-1', 'REQ-2'].

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the `collectors` module's ability to handle ReprFileLocation in error reports.

Why Needed: This test prevents a potential crash when encountering ReprFileLocation in error reports, ensuring the collector can recover and continue processing other errors.

Key Assertions:

- The `_extract_error` method of `TestCollector` should return 'Crash report' when called with an instance of `ReprFileLocation`.
- The `longrepr` attribute of `Report` should have a value that is a string representation of 'Crash report'.
- The `__str__` method of `Report.longrepr` should be able to return the expected string 'Crash report' when called. If this assertion fails, it may indicate a bug in the collector or its dependencies.
- If an instance of `ReprFileLocation` is passed to `_extract_error`, it should not cause a crash report.
- The `collectors` module should be able to recover from errors and continue processing other errors without crashing.
- If an error occurs during the collection process, the test should fail with a meaningful error message.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the `_extract_error` method returns a string 'longrepr' when called with a `report` object containing a 'longrepr' attribute.

Why Needed: This test prevents a potential regression where the collector might not correctly extract error strings from reports.

Key Assertions:

- The `report.longrepr` attribute is set to 'Some error occurred'.
- The `_extract_error` method returns 'Some error occurred'.
- The extracted string matches the expected value.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the test fails if there are no longreprs to extract skip reasons from.

Key Assertions:

- The `_extract_skip_reason` method does not raise an exception or return a specific value when `report.longrepr` is `None`.
- The `_extract_skip_reason` method returns `None` instead of raising an exception or returning a default value when `report.longrepr` is `None`.
- The `_extract_skip_reason` method checks for the existence of `report.longrepr` before attempting to extract skip reasons and raises an exception if it does not exist.
- When `report.longrepr` is `None`, the `_extract_skip_reason` method correctly returns `None` without raising an exception or returning a default value.
- The `_extract_skip_reason` method checks for the existence of `report.longrepr` before attempting to extract skip reasons and raises an exception if it does not exist.
- When `report.longrepr` is `None`, the `_extract_skip_reason` method correctly returns `None` without raising an exception or returning a default value.
- The `_extract_skip_reason` method checks for the existence of `report.longrepr` before attempting to extract skip reasons and raises an exception if it does not exist.
- When `report.longrepr` is `None`, the `_extract_skip_reason` method correctly returns `None` without raising an exception or returning a default value.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test `test_extract_skip_reason_string` verifies that the `_extract_skip_reason` method returns a string as expected.

Why Needed: This test prevents potential issues where the method does not return the correct skip reason string, potentially leading to incorrect reporting or debugging.

Key Assertions:

- The method should return 'Just skipped' as the skip reason string.
- The method should extract the `longrepr` attribute from the report object and return it as a string.
- The method should not raise an exception if no longrepr is available.
- The method should handle cases where the report object does not have a `longrepr` attribute.
- The method should not modify the original report object.
- The method should preserve the original type and behavior of the report object.
- The method should return a string that can be used as a valid reason for skipping in the collector.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that extract skip reason tuple works correctly.

Why Needed: Prevents a potential bug where the test fails due to incorrect handling of tuples with longrepr messages.

Key Assertions:

- The `report.longrepr` attribute should contain a tuple with `(file, line, message)` as its first element.
- The `message` field within this tuple should be 'Skipped for reason'.
- The `longrepr` value should match the expected string when converted to a Python literal.
- The `report.longrepr` attribute should contain the entire tuple as its second element.
- The `message` field within this tuple should be 'Skipped for reason'.
- The `longrepr` value should match the expected string when converted to a Python literal.
- The `report.longrepr` attribute should contain the entire tuple as its second element.
- The `message` field within this tuple should be 'Skipped for reason'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: When the `handle_collection_report` method is called with a report that indicates collection failure, it should record this error.

Why Needed: This test prevents potential data loss due to unhandled collection errors.

Key Assertions:

- The length of `collector.collection_errors` should be equal to 1.
- The nodeid in the first `collector.collection_errors` item should match 'test_broken.py'.
- The message in the first `collector.collection_errors` item should match 'SyntaxError'.

COVERAGE

src/pytest_llm_report/collector.py	21 lines (ranges: 58, 60-65, 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- res.rerun_count should be equal to 1 (the expected number of reruns).
- res.final_outcome should be 'failed' (indicating that the test failed after rerunning).

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: TestCollectorReportHandling::test_handle_runtest_setup_failure verifies that a setup error is recorded in the report.

Why Needed: This test prevents regression where TestCollector handles runtest log reports as if they were successful, potentially leading to incorrect reporting of setup errors.

Key Assertions:

- res.outcome should be 'error'
- res.phase should be 'setup'
- res.error_message should be 'Setup failed'

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test case 'Should record error if teardown fails after pass' verifies that the collector correctly records an error when a teardown operation fails.

Why Needed: This test prevents regression by ensuring that the collector handles teardown failures properly and reports them as errors in the results.

Key Assertions:

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

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 _parse_preferred_models method with edge cases to ensure correct behavior.

Why Needed: This test prevents regression in case a new model is added to the preferred list without updating the parsing logic.

Key Assertions:

- The function should return an empty list when the 'model' parameter is None.
- The function should return an empty list when the 'model' parameter is set to 'All'.
- The function should not throw any errors or exceptions when the 'model' parameter is None and the preferred models are already empty.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the rate limiter does not allow excessive tokens when there are available tokens but no requests.

Why Needed: This test prevents a bug where the rate limiter allows too many tokens when there are enough available tokens, potentially leading to unexpected behavior or errors.

Key Assertions:

- assert limiter.next_available_in(60) > 0
- assert limiter.next_available_in(10) == 0
- assert limiter.record_tokens(50) < 100
- assert len(limiter.tokens) > 0

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/gemini.py`	35 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 81-82, 84, 87-88, 92-93, 95-96, 100-101, 103, 105, 107-114)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the `to_dict()` method of `SourceCoverageEntry` and `LlmAnnotation` correctly returns their respective values.

Why Needed: This test prevents regression in coverage booster models where the coverage percentage is not being accurately calculated.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: The `CoverageMapper` instance should be initialized with the provided configuration.

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

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 in the `CoverageMapper` class should be able to retrieve a list of warnings from the coverage report.

Why Needed: This test prevents potential issues where the function returns an incorrect type (in this case, a list) when it's expected to return a specific type (a list).

Key Assertions:

- The `get_warnings` method is called on an instance of `CoverageMapper`.
- A `Config` object is created and passed to the `CoverageMapper` instance.
- The `get_warnings` method is called on the `CoverageMapper` instance.
- The result of the `get_warnings` method is checked to be a list.
- The type of the result is checked to be a list.
- A warning message or error is expected to be returned from the `get_warnings` 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`	`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: Tests coverage map function with no coverage file.

Why Needed: Prevents a potential bug where the function returns an empty dictionary when there is no coverage file.

Key Assertions:

- The `Path.exists` mock returns False.
- The `glob.glob` mock returns an empty list.
- The `map_coverage` function should return an empty dictionary.
- There should be at least one warning in the `warnings` list.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the `CoverageMapper` extracts all phases when `include_phase=all`.

Why Needed: This test prevents a regression where the coverage is not extracted for all phases when `include_phase=all`.

Key Assertions:

- The method `_extract_nodeid` should return the correct node ID for each phase.
- The method `_extract_nodeid` should handle cases where the phase name contains multiple words (e.g., 'test_foo|run').
- The method `_extract_nodeid` should correctly extract the node ID for phases that are not explicitly mentioned in the test code (e.g., `test.py::test_foo|teardown`).

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the `extract_nodeid` method returns `None` when given an empty context.

Why Needed: This test prevents a potential bug where the `extract_nodeid` method does not handle empty contexts correctly, potentially leading to incorrect coverage metrics or unexpected behavior.

Key Assertions:

- The `_extract_nodeid` method should return `None` for an empty string.
- The `_extract_nodeid` method should return `None` for a `None` value as context.
- The test should fail when given an empty context, indicating that the `extract_nodeid` method is not handling it correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Test the function `extract_nodeid_filters_setup` to ensure it correctly filters out setup phase when `include_phase` is set to 'run'.

Why Needed: This test prevents a potential bug where the function does not filter out setup phase, potentially leading to incorrect coverage analysis.

Key Assertions:

- The function `_extract_nodeid` should return None for node IDs that start with `test.py::test_foo|setup`.
- The function `_extract_nodeid` should raise an error when called with a non-string 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/coverage_map.py	9 lines (ranges: 44-45, 216, 220, 224-225, 228-230)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the `extract_nodeid` method extracts the correct node ID from a run phase context.

Why Needed: This test prevents regression by ensuring that the node ID extraction logic is correct and consistent across different phases.

Key Assertions:

- The extracted node ID matches the expected value (`test.py::test_foo`) for the given module and function name.
- The extracted node ID does not contain any leading or trailing whitespace.
- The extracted node ID does not contain any invalid characters (e.g., special regex patterns).
- The extracted node ID is present in the `run` phase context.
- The extracted node ID is present for all modules and functions within the given module.
- No exceptions are raised when extracting the node ID from a run phase 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	11 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test should extract contexts for full logic coverage of _extract_contexts method.

Why Needed: This test prevents regression that would occur if the _extract_contexts method did not cover all paths in _extract_contexts.

Key Assertions:

- mocked context_by_lineno returns a dictionary with test_one and test_two as keys, where each key has multiple values.
- the function should return a list of files that have app.py as their file path.
- the line count for the returned files in test_one should be 2 (lines 1 and 2).
- the line count for the returned files in test_two should not be affected by the mock data.
- mocked contexts_by_lineno does not return any additional context files that are not already included in the result.
- the function should correctly handle cases where there are multiple lines of code with the same file path (e.g., app.py::test_one|run).
- the function should correctly handle cases where a file has no matching contexts (e.g., README.md::test_three|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	57 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 137-140, 144, 148, 150, 152-153, 156, 160-163, 165, 167-168, 173, 176, 178-184, 187-189, 191-194, 196, 199-200, 202, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	17 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64, 100, 103, 111-112, 116, 123)
src/pytest_llm_report/util/ranges.py	13 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65-67)

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

AI ASSESSMENT

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

Why Needed: Prevents a regression where the coverage map is incorrectly populated with context information for files without any test contexts.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the `CoverageMapper` extracts node IDs for tests with missing lines in different phases.

Why Needed: This test prevents a bug where the `CoverageMapper` incorrectly filters out node IDs from tests with missing lines in certain phases.

Key Assertions:

- The `_extract_nodeid` method returns the expected node ID for each test.
- The `_extract_nodeid` method correctly ignores node IDs from tests with missing lines in certain phases.
- The `CoverageMapper` uses the correct phase to filter out node IDs from tests.
- The `CoverageMapper` ignores node IDs from tests without a specified phase.
- The `CoverageMapper` does not incorrectly report node IDs for tests with missing lines in different phases.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the test_load_coverage_data_no_files function correctly handles the case when no coverage files exist.

Why Needed: This test prevents a potential bug where the test would fail due to missing coverage data.

Key Assertions:

- The function should return None for _load_coverage_data() without any .coverage files.
- There should be exactly one warning message with code 'W001' when no .coverage files exist.
- The warnings list should contain only one item with the specified 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`	`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 should handle errors reading coverage files with a corrupt .coverage file.

Why Needed: This test prevents regression by ensuring that the CoverageMapper can correctly handle corrupted coverage data.

Key Assertions:

- The function _load_coverage_data() returns None when an error occurs during read.
- Any warnings generated by mapper.warnings contain the message 'Failed to read coverage data'.
- The function _load_coverage_data() raises an Exception with the message 'Corrupt coverage file' when a corrupt .coverage file is encountered.

COVERAGE

`src/pytest_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` 2ms 🛡️ 4

AI ASSESSMENT

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

Why Needed: This test prevents regression in the CoverageMapper class, which is responsible for handling parallel coverage files from xdist. Without this test, the mapper may not update its internal data structures correctly when dealing with parallel files, leading to incorrect coverage data.

Key Assertions:

- The `update` method of the `CoverageData` instance should be called at least twice during the `_load_coverage_data` process.
- The `update` method of the `CoverageData` instance should not be called more than once during the `_load_coverage_data` process.
- The number of times the `update` method is called for each mock CoverageData instance should match the expected number of parallel coverage files.
- The `update` method should only be called when a new parallel coverage file is created and removed from the temporary directory.
- The `update` method should not be called when the existing parallel coverage file is updated or deleted in the temporary directory.
- The `update` method should call the original mock CoverageData instance's `update` method for each mock instance.
- The `update` method should update the internal data structures of the CoverageMapper correctly after calling its side effect instances.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the `map_coverage` method returns an empty dictionary when `_load_coverage_data` returns None.

Why Needed: Prevents a potential bug where the test fails due to a missing coverage map being returned from `_load_coverage_data`.

Key Assertions:

- The function should return an empty dictionary when no data is loaded.
- The function should not raise any exceptions when no data is loaded.
- The function should handle the case where `None` is returned by `_load_coverage_data` correctly.
- The function should preserve the original coverage map values.
- The function should ignore missing keys in the coverage map.
- The function should return a dictionary with default values (e.g., `{}`) when no data is loaded.
- The function should not throw an exception when `None` is returned by `_load_coverage_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/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 coverage map source coverage analysis error.

Why Needed: Prevents test failure due to analysis errors during source coverage mapping.

Key Assertions:

- The mock analysis2 function should be called with an Exception exception.
- The mocked mock_data.measured_files.return_value should return ['app.py'].
- The mock_cov.get_data.return_value should raise an Exception exception.
- The entries list should have zero length after skipping files with errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	22 lines (ranges: 44-45, 243-244, 246-248, 250, 252-254, 259, 261, 263-268, 271, 299-300)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116, 123)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function `entries[0].file_path` should return the path of the source file being tested (`'app.py'`).
- The function `entries[0].statements` should return the number of statements in the source file (`3`).
- The function `entries[0].covered` should return the percentage of covered lines (`2`).
- The function `entries[0].missed` should return the number of missed lines (`1`).
- The function `entries[0].coverage_percent` should return the percentage of covered lines (`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 correctly returns a WarningCode.W001_NO_COVERAGE instance with the specified detail.

Why Needed: To prevent a bug where an unknown warning is returned without any additional information.

Key Assertions:

- The `message` attribute of the returned WarningCode.W001_NO_COVERAGE instance contains the expected string 'No .coverage file found'.
- The `detail` attribute of the returned WarningCode.W001_NO_COVERAGE instance matches the specified value 'test-detail'.
- The `code` attribute of the returned WarningCode.W001_NO_COVERAGE instance is set to WarningCode.W001_NO_COVERAGE.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that warning codes have correct values.

Why Needed: This test prevents a potential regression where the warning code values are not correctly set.

Key Assertions:

- {'message': 'Assertion error', 'value': 'W001'}
- {'message': 'Assertion error', 'value': 'W101'}
- {'message': 'Assertion error', 'value': 'W201'}
- {'message': 'Assertion error', 'value': 'W301'}
- {'message': 'Assertion error', '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 warning to dict method.

Why Needed: Prevent a potential bug where Warning objects are not properly converted to dictionaries.

Key Assertions:

- The 'code' key should be present in the dictionary and have the correct value.
- The 'message' key should also be present in the dictionary with the correct value.
- The 'detail' key should be present in the dictionary if it exists, otherwise its value should be an empty string.
- All keys in the dictionary should match the expected keys according to WarningCode.
- If a Warning object has no detail, then the 'message' and 'code' keys should both be missing 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/errors.py`	6 lines (ranges: 70-72, 74-76)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the warning is not correctly generated for known code.

Key Assertions:

- The function `make_warning` should return an instance of `WarningCode.W101_LLM_ENABLED` with the correct message.
- The warning message should be set to `WARNING_MESSAGES[WarningCode.W101_LLM_ENABLED]`.
- The detail attribute should be set to `None` for this specific 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/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 MakeWarningUnknownCode verifies that missing WarningCode.W001_NO_COVERAGE is handled correctly.

Why Needed: This test prevents a bug where the fallback message for unknown code is not used when an enum is allowed in the typed function.

Key Assertions:

- The expected message 'Unknown warning.' is returned when making a warning with WarningCode.W001_NO_COVERAGE.
- The old message 'Warning: Unknown warning.' is restored after the test.
- The WARNING_MESSAGES dictionary is updated correctly to reflect the new fallback 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/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 makes a warning when invalid configuration is provided with detail.

Why Needed: Prevents a potential bug where the test does not create a warning for an invalid configuration with detailed error message.

Key Assertions:

- w.code == WarningCode.W301_INVALID_CONFIG
- w.detail == 'Bad value'
- assert w is not None

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that all WarningCode enum values are strings.

Why Needed: This test prevents a potential bug where the WarningCode enum is not properly initialized with string values.

Key Assertions:

- code.value should be an instance of str.
- code.value should start with 'W'.
- All WarningCode enum members should have a value that starts with 'W' and is a string.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that Warning.to_dict() returns a dictionary without 'detail' key when no detail is provided.

Why Needed: This test prevents the warning 'WarningDataClass: warning_to_dict_no_detail' because it ensures that the Warning object's attributes are serialized correctly to a dictionary without including any additional details.

Key Assertions:

- The 'code' attribute of the Warning object is set to 'W001'.
- The 'message' attribute of the Warning object is set to 'No coverage'.
- The 'detail' key is not present in the serialized dictionary.
- The length of the serialized dictionary is 2 (i.e., it only contains 'code' and 'message').

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test the warning to dictionary conversion with detail.

Why Needed: This test prevents a potential bug where warnings are not properly serialized to dictionaries, potentially causing issues in downstream data processing or logging.

Key Assertions:

- The `to_dict()` method of the Warning class returns a dictionary with the correct keys and values.
- The 'code' key in the dictionary contains the warning code.
- The 'message' key in the dictionary contains the warning message.
- The 'detail' key in the dictionary contains the detailed warning message.
- The 'WarningCode.W001_NO_COVERAGE' value is correctly assigned to the 'code' key.
- The 'Check setup' value is correctly assigned to the 'detail' key.
- The resulting dictionary has the correct structure and content.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/fs.py	1 lines (ranges: 79)

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

AI ASSESSMENT

Scenario: Verifies that the `is_python_file` function 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 path to a `.py` file.
- The function should raise an error or return a specific value when given a non-`.py` file path.
- The function should correctly handle relative paths for `.py` files.
- The function should not incorrectly identify other types of files as Python files.
- The function should be able to handle multiple extension checks (e.g., `.txt`, `.json`) in a single call.
- The function should raise an error when given a file with a different encoding than UTF-8.
- The function should correctly handle cases where the file is empty or contains only whitespace.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/fs.py`	1 lines (ranges: 79)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression when creating files outside of the test directory.

Key Assertions:

- The file is created with the correct parent directory.
- The file's path is not the same as its original path.
- The file is written to the correct location (subdir/file.py).
- The file's path is absolute (i.e., it does not start with a slash).
- The file's parent directory exists and can be created without raising an error.
- The file's parent directory is not already in the test directory.
- The file's original path is not the same as its relative path.
- The file's relative path starts with 'subdir/'.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents a potential issue where an absolute path would be returned instead of a relative one.

Key Assertions:

- The result of `make_relative('foo/bar')` should be 'foo/bar'.
- The function does not return the same result when given an absolute path like 'foo/absolute/path'.
- If the input is a directory, the function returns the relative path to that directory.
- If the input is a file, the function returns the relative path from the current working directory.
- The function handles cases where the base directory is empty or None.
- The function does not return an error when given an invalid input (e.g., non-string base)
- The function preserves the original case of the input path (e.g., 'foo' instead of 'Foo').

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/fs.py`	7 lines (ranges: 30, 33, 36, 39, 42, 55-56)

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

AI ASSESSMENT

Scenario: Tests that a normalized path is returned for an already-normalized input.

Why Needed: This test prevents a potential bug where the `normalize_path` function would incorrectly return the original path if it's already normalized.

Key Assertions:

- The `normalize_path` function should return the same result as calling `pathlib.PurePath('foo/bar')`.
- The `normalize_path` function should not modify the input path.
- The `normalize_path` function should handle paths with leading or trailing slashes correctly.
- The `normalize_path` function should preserve the original directory hierarchy.
- The `normalize_path` function should raise an error if the input is not a string or a Path object.
- The `normalize_path` function should return an empty string for an empty path.
- The `normalize_path` function should handle symbolic links 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::TestNormalizePath::test_forward_slashes` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `normalize_path` function correctly converts forward slashes in file paths to forward slashes.

Why Needed: This test prevents a potential bug where the function does not handle forward slashes correctly, potentially leading to incorrect path comparisons or errors.

Key Assertions:

- The function should convert '\"' to '/\'.
- The function should convert '/foo/bar' to 'foo/bar'.
- The function should preserve the original directory structure and only replace forward slashes with 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/test_fs.py::TestNormalizePath::test_strips_trailing_slash

Why Needed: This test prevents a potential bug where the function does not correctly handle paths with trailing slashes.

Key Assertions:

- The input path should be stripped of any trailing slash before normalization.
- Normalization should return the same path if it already did not have a trailing slash.
- The function should raise an error when given a path with no leading directory or file name.
- The function should correctly handle paths like 'foo/../bar/' and 'foo/bar/'.
- Normalization of empty strings should return an empty string.
- Normalization of absolute paths (like '/home/user/foo/') should work as expected.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/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 that the `should_skip_path` function correctly skips custom paths based on provided patterns.

Why Needed: This test prevents a potential regression where the function does not skip custom paths as intended, potentially leading to unexpected behavior or errors.

Key Assertions:

- custom pattern 'test*' should be matched and the function should return True for `tests/conftest.py`
- custom pattern 'test*' should not match and the function should return False for `src/module.py`

COVERAGE

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

Key Assertions:

- assert should_skip_path('src/module.py') == False
- assert not should_skip_path('non-existent-path.txt') == True

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the function incorrectly skips non-`.git` directories, potentially leading to unexpected behavior or errors.

Key Assertions:

- assert should_skip_path('.git/objects/foo') is True
- assert should_skip_path('non_git_directory') is False

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents a potential issue where the test would incorrectly skip non-pycache directories.

Key Assertions:

- The function should return True for directories with a name starting with '__pycache__' and containing '.pyc'.
- The function should not return True for directories without a name starting with '__pycache__' or containing '.pyc'.
- The function should correctly handle cases where the directory name is not exactly 'foo/__pycache__/bar.pyc'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 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 `should_skip_path` function should return True for venv directories and False for regular Python libraries.
- The `should_skip_path` function should not return True for `.venv` directories or any other directory that is a subdirectory of the current working directory.
- The test should verify that the `should_skip_path` function correctly handles different types of Python library directories.
- The test should check that the `should_skip_path` function does not incorrectly identify venv directories as being to be skipped in certain scenarios.
- The test should ensure that the `should_skip_path` function is able to handle nested directory structures correctly.
- The test should verify that the `should_skip_path` function returns False for all other Python library directories.
- The test should check that the `should_skip_path` function does not return True for any other type of directory that is a subdirectory of the current working directory.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where the rate limiter incorrectly clears request and token usage counts for requests made in the past.

Key Assertions:

- The length of _request_times should be greater than 0 after calling _prune()
- The length of _token_usage should be greater than 0 after calling _prune()

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents requests from exceeding a certain threshold (in this case, 1 RPM)

Why Needed: This test prevents a potential issue where multiple requests are made within a short time frame and exceed the rate limit, causing the API to become unavailable.

Key Assertions:

- The `next_available_in` method returns a value greater than 0
- The `next_available_in` method returns a value less than or equal to 60.0

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test verifies that the rate limiter correctly handles cases where tokens are not yet available, preventing potential regressions.

Key Assertions:

- The next_available_in method should return a value greater than 0 after waiting for 10 tokens.
- The _token_usage list should contain exactly two elements when the last token is recorded.
- The limiter._token_usage list should not be empty before and after recording the second token.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/gemini.py`	`33 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 81-82, 84, 87-88, 92-94, 100-101, 103, 105, 107-108, 110-114)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

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

AI ASSESSMENT

Scenario: Verify that the `wait_for_slot` method of `_GeminiRateLimiter` sleeps when a request is made.

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

Key Assertions:

- The `wait_for_slot` method should be called with a mock `time.sleep` function.
- The `wait_for_slot` method should assert that it was called with the correct number of arguments (1).
- The `wait_for_slot` method should not call `mock_sleep` immediately, but instead after waiting for the specified amount of time.
- The `wait_for_slot` method should not return any value.
- The `wait_for_slot` method should raise an exception if it is called with a negative number of arguments (requests per minute).
- The `wait_for_slot` method should raise an exception if it is called with a non-integer argument (number of requests 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	31 lines (ranges: 39-42, 45-46, 48, 52-54, 58-59, 61-63, 73, 76-78, 81-82, 84, 87-88, 92-93, 95, 97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the rate limiter does not record tokens even when there are none available.

Key Assertions:

- The `_token_usage` list of the rate limiter should be empty after calling `record_tokens(0)`.
- The length of `_token_usage` should be 0.
- No exception should be raised when no tokens are available for recording.
- The rate limiter's internal state should reflect that no tokens were recorded.
- _token_usage should not contain any token usage data.
- The rate limiter's `tokens_per_minute` attribute should still be valid and unchanged.
- The `_GeminiRateLimiter` instance should still have a valid `limits` object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: Test the rate limiter to prevent exceeding daily requests.

Why Needed: This test prevents a potential bug where the rate limiter is exceeded by more than one request per day, causing an error.

Key Assertions:

- The function `wait_for_slot` should raise `_GeminiRateLimitExceeded` with the correct message 'requests_per_day'.
- The function `record_request` does not return any value.
- The function `wait_for_slot` should wait for a slot to become available after each request, and raise an error if no slots are available within the specified time frame.
- _GeminiRateLimitExceeded is raised with the correct message 'requests_per_day'.
- The rate limiter's daily limit is correctly checked before allowing another request.
- The function `wait_for_slot` does not wait for a slot to become available after each request, and raises an error if no slots are available within the specified time frame.
- _GeminiRateLimitExceeded is raised with the correct message 'requests_per_day'.
- The rate limiter's daily limit is correctly checked before allowing another request, and the function `wait_for_slot` waits for a slot to become available after each request.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify the test_gemini_limiter_tpm_fallback_wait function covers the case when TPM wait time fallback occurs.

Why Needed: This test prevents a potential regression where the rate limiter fails to detect and handle cases when the TPM is not available for a long enough period, leading to unexpected behavior or errors in the application.

Key Assertions:

- The function `_GeminiRateLimitConfig(tokens_per_minute=10)` sets up the correct rate limit configuration for the test environment.
- The `limiter.record_tokens(10)` call fills up the TPM with tokens as expected.
- The calculation of `wait = limiter._seconds_until_tpm_available(now, 5)` returns a non-zero value indicating that the TPM is not available within the specified time frame.
- The assertion `assert wait > 0` ensures that the function waits for at least some time before checking if the TPM is available again.
- The line `# Line 116 hit because tokens_used + request_tokens > limit AND token_usage is not empty` verifies that the rate limiter correctly detects and handles cases where the TPM is not available.
- The expected behavior of filling up the TPM with tokens as described in the test scenario is verified through the 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/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 614ms 🛡️ 6

AI ASSESSMENT

Scenario: Test that RPM rate limit cooldown handling is properly enforced.

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

Key Assertions:

- The 'models/gemini-pro' model should be in the cooldowns dictionary with a value greater than 1000.0 seconds.
- The cooldown value for 'models/gemini-pro' should be greater than 1000.0 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	23 lines (ranges: 52-53, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	117 lines (ranges: 32-34, 39-42, 45-46, 48, 52-54, 66, 68-70, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-215, 217-223, 225-227, 233-234, 238-240, 242-243, 274-277, 280, 282-290, 292-295, 297-298, 300-301, 346, 348-350, 352-353, 381-382, 385-386)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression when the provider encounters a rate limit and retries after a certain delay.

Key Assertions:

- The annotation should have the correct scenario 'Recovered Scenario',
- The mock post call count should be 2 (1 for the first failed request, 1 for the second successful one),
- The provider's _parse_response method should return a Mock object with the expected scenario and error.
- The provider's _annotate_internal method should not raise an exception when encountering a rate limit retry scenario.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that _annotate_internal returns the correct LlmAnnotation for a successful annotation.

Why Needed: This test prevents regression where _parse_response might expect an incorrect format of response from _call_gemini.

Key Assertions:

- The scenario of 'Success Scenario' is correctly extracted from the annotation.
- The error in the annotation is None.
- The annotation does not contain any errors.
- The annotation's scenario matches the expected value.
- The annotation does not contain any invalid or unexpected data.
- The _parse_response function returns a Mock object with the correct scenario and no error.
- The _annotate_internal function correctly calls _parse_response to extract the annotation.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	2 lines (ranges: 52-53)
`src/pytest_llm_report/llm/gemini.py`	173 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-223, 225-227, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-343, 346, 348-350, 352-355, 360-363, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verifies that the `GeminiProvider` class correctly checks availability based on environment variables.

Why Needed: This test prevents a potential bug where the `GeminiProvider` class does not handle environment variable changes properly, potentially causing unexpected behavior or errors.

Key Assertions:

- The `_check_availability()` method of the `GeminiProvider` class should return `False` when no environment variables are set.
- The `_check_availability()` method of the `GeminiProvider` class should return `True` when environment variable `GEMINI_API_TOKEN` is set to a valid value.
- Environment variable changes should not affect the availability check result of the `GeminiProvider` class.
- Setting `GEMINI_API_TOKEN` environment variable before creating an instance of `GeminiProvider` should not change its availability check result.
- Creating an instance of `GeminiProvider` with a different provider name (`gemini`) should not affect its availability check 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/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	10 lines (ranges: 134, 136-139, 141-142, 266-267, 269)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- The next_available_in method returns None after 100 requests have been recorded.
- The rate limiter does not allow any further requests to be processed until the daily limit is reached again (i.e., 101st request).
- The rate limiter prevents more than one request from being processed within a single day, ensuring consistent and predictable behavior.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the rate limiter does not block subsequent requests after the first two have passed.

Why Needed: This test prevents a potential bug where subsequent requests are blocked due to insufficient available time.

Key Assertions:

- The next_available_in method should return 0.0 for the first two requests.
- The next_available_in method should return 0.0 after recording the third request.
- The wait value should be greater than 0 and less than or equal to 60.0 seconds.
- _GeminiRateLimitConfig(requests_per_minute=2) should not have been called before the first two requests.
- _GeminiRateLimiter(limits) should have created a rate limiter instance with the specified 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/gemini.py`	27 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-97, 100-102)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that different configurations produce different hashes.

Why Needed: This test prevents a potential bug where the same configuration produces the same hash, potentially leading to inconsistencies in data storage or retrieval.

Key Assertions:

- The function `compute_config_hash` should return a different value for two different configurations.
- The hash of `config1` should not be equal to the hash of `config2`.
- The hash of `config1` should not be equal to the hash of `Config(provider='ollama')` (the second config is created with provider 'ollama' but its actual provider is still 'none').
- The hash of `Config(provider='ollama')` should not be equal to the hash of `Config(provider='none')` (the first config is created with provider 'none' but its actual provider is still 'ollama').

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The length of the computed hash should be exactly 16 characters.
- The hash value should not exceed 15 characters to prevent truncation errors.
- The hash value should start with '0x' prefix if it's a hexadecimal string.
- The hash value should contain only alphanumeric characters and underscores.
- No leading zeros are allowed in the hash value.
- The hash value should be at least 16 characters long but no more than 32 characters long.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the computed SHA-256 hash of a file matches its content hash.

Why Needed: Prevents regression where the file's content changes but the file hash remains the same.

Key Assertions:

- The computed SHA-256 hash of the file should be equal to the content hash.
- The content hash of the file should match the expected value.
- The file hash should not change even if the file's content is modified.
- The file hash should remain consistent across different runs of the test
- The computed SHA-256 hash of a file with a different content but same path should be equal to the content hash.
- The content hash of a file with a different content but same path and different name should also be equal to 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: Test the hashing function on a file with known contents.

Why Needed: Prevents a potential bug where the hashing function fails to correctly hash files with non-ASCII characters or other special cases.

Key Assertions:

- The length of the computed SHA256 hash should be 64 bytes.
- The first byte of the hash should match 'hello'.
- The second byte of the hash should match 'world'.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the same key is used for multiple computations, potentially leading to unexpected signature differences.

Key Assertions:

- Verify that two different keys produce distinct HMAC signatures for the same input.
- Check if the computed signature for 'key1' is not equal to the computed signature for 'key2'.
- Ensure the first key produces a unique signature, and the second key does not.
- Verify the difference in signatures between 'key1' and 'key2'.
- Confirm that the order of keys does not affect the generated signature.
- Test if using different keys results in different HMAC values for the same input.
- Analyze the behavior when multiple keys are used for a single computation.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify the length of the HMAC signature for a given content and secret key.

Why Needed: This test prevents a potential issue where an attacker could exploit the length of the HMAC signature to deduce the secret key.

Key Assertions:

- The length of the HMAC signature should be 64 bytes.
- The length of the HMAC signature is not less than 32 bytes.
- The length of the HMAC signature is not greater than 128 bytes.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the SHA-256 hash of the same input produces the same output.

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

Key Assertions:

- The function `compute_sha256(b'...'` should return the same hash for two identical bytes objects.
- The function `compute_sha256(b'...'` should raise an exception if the input is not a bytes object.
- The function `compute_sha256(b'...'` should produce the same hash as the built-in `hash()` function for the input.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The hash function should produce a SHA-256 hash of the input string "test" which is expected to have a length of 64 characters.

Why Needed: This test prevents a potential bug where the hash length is not as expected due to incorrect implementation or configuration of the compute_sha256 function.

Key Assertions:

- The output of the compute_sha256 function should be a bytes object containing 64 hexadecimal characters.
- The hexadecimal representation of the hash should match the expected value (e.g., '48656c6c6f20576f726c64')
- The length of the resulting string should be exactly 64 characters
- The output string should not contain any null bytes () or other non-hexadecimal characters
- The hexadecimal representation of the hash should have a total length of 64 characters (16 bytes)
- The hash should be a valid SHA-256 hash (e.g., it should start with '48656c6c6f20576f726c64')

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 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` 67ms 🛡️ 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 dependency snapshot.
- The presence of 'pytest' in the dependency snapshot indicates that it is available for installation.
- Including 'pytest' in the dependency snapshot ensures its availability for testing purposes.
- The absence of 'pytest' in the dependency snapshot may indicate a missing or outdated package.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 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` 73ms 🛡️ 3

AI ASSESSMENT

Scenario: The `get_dependency_snapshot()` function should return a dictionary.

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

Key Assertions:

- snapshot is indeed a dictionary.
- the 'get_dependency_snapshot()' function is called and its result is stored in the `snapshot` variable.
- assert isinstance(snapshot, dict) is used to verify that the returned value matches the expected type.
- if snapshot were not a dictionary, this test would fail.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test loads HMAC key from file.

Why Needed: Prevents a potential bug where the loaded key is not correctly decoded from the file.

Key Assertions:

- The `load_hmac_key` function should return the expected value of the HMAC key.
- The `load_hmac_key` function should handle cases where the file contents are not valid JSON.
- The `load_hmac_key` function should raise an error if the file does not exist or is not a valid file.
- The `load_hmac_key` function should correctly decode the HMAC key from the file even if it's missing the expected header.
- The `load_hmac_key` function should handle cases where the file contains multiple keys.
- The `load_hmac_key` function should ignore any extra data in the file that is not related to the HMAC key.
- The `load_hmac_key` function should correctly handle cases where the file has a different encoding than UTF-8.
- The `load_hmac_key` function should raise an error if the file contains invalid JSON.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 'test_missing_key_file' verifies that the function returns None when a missing key file is provided.

Why Needed: This test prevents a potential bug where the function fails to return an expected result (None) when a key file does not exist.

Key Assertions:

- The function should return `None` when a key file with the specified path (`/nonexistent.key`) is provided.
- The function should raise a `KeyError` exception if the key file exists but cannot be loaded due to permission issues or other reasons.
- The test should verify that the function correctly handles cases where the key file does not exist, without attempting to load it.
- The function's return value should match the expected result (None) in all scenarios.
- The test should cover different paths for the missing key file (e.g., `/nonexistent.key`, `/nonexistent2.key`)
- The function's behavior should be consistent across different Python versions and platforms
- The test should report an error message indicating that the key file was not found, rather than raising a cryptic 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/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/hashing.py`	4 lines (ranges: 73, 76-78)

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

AI ASSESSMENT

Scenario: Test that the function returns `None` when no key file is provided.

Why Needed: Prevents a potential bug where the function does not handle the case when no key file is configured.

Key Assertions:

- The function should return `None` without attempting to load any HMAC keys.
- No error or exception should be raised when loading an empty configuration.
- The function's behavior should be consistent across different test environments.
- No exceptions should be thrown when calling the `load_hmac_key` method with a `Config` object that does not contain an HMAC key.
- The function's return value should be `None` instead of raising an exception or returning an error message.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	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 that aggregation defaults are set correctly.

Why Needed: This test prevents a potential bug where the default aggregation policy is not applied correctly.

Key Assertions:

- config.aggregate_dir should be None.
- config.aggregate_policy should be 'latest'.
- config.aggregate_include_history 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`	`3 lines (ranges: 107, 147, 233)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

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

AI ASSESSMENT

Scenario: Verify that the `capture_failed_output` default value is set to `False` for the integration gate.

Why Needed: This test prevents a potential bug where the `capture_failed_output` default value might be incorrectly set to `True`, leading to unexpected behavior in the integration tests.

Key Assertions:

- The `config.capture_failed_output` attribute is not equal to `False`.
- The `config.capture_failed_output` attribute is not equal to `None`.
- The `capture_failed_output` default value is set to `False`.
- The `capture_failed_output` default value does not match the expected behavior in all test environments.
- The `capture_failed_output` default value is not overridden by any environment variables or configuration 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	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Tests the default context mode for integration gate.

Why Needed: Prevents a potential bug where the context mode is not set to 'minimal' by default.

Key Assertions:

- The function `get_default_config()` returns an instance of `Config` with an `llm_context_mode` attribute equal to 'minimal'.
- The value of `config.llm_context_mode` is indeed 'minimal'.
- If the context mode is not set to 'minimal', a different default configuration would be used.
- Setting the context mode to another value (e.g., 'default') would prevent this test from passing.
- The `llm_context_mode` attribute of the `Config` instance is not changed by setting it to an invalid value.
- If the context mode is set to a valid value, such as 'minimal', the function returns the expected result.
- A different configuration with a different context mode would be used if the default is not set correctly.
- The test would fail if the `llm_context_mode` attribute of the `Config` instance is changed after setting it to an invalid value.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: The test prevents a potential bug where the LLM is enabled by default, potentially causing unexpected behavior or errors.

Key Assertions:

- config.is_llm_enabled() == False
- config.get_llm_enabled_value() == False
- get_default_config().llm_enabled() == False
- get_default_config().get_llm_enabled_value() == False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 `get_default_config()` function returns a configuration object with an `omit_tests_from_coverage` attribute set to `True`.

Why Needed: This test prevents a regression where the default configuration does not include tests by omission.

Key Assertions:

- The `config` variable is of type `dict` and has an `omit_tests_from_coverage` key with value `True`.
- The `config` dictionary contains the expected keys: `omit_tests_from_coverage`.
- The `config` dictionary does not have any other keys that could be causing issues.
- The `config` dictionary is a valid Python object.
- The `get_default_config()` function returns a configuration object with an `omit_tests_from_coverage` attribute set to `True`.
- The `config` variable has the expected value for the `omit_tests_from_coverage` key.
- The test does not fail when the default configuration includes tests by omission.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- config.provider should be equal to 'none'
- config.provider should not be equal to any other value
- config.provider should have a default value of 'none'
- get_default_config() should return an object with provider set to 'none'
- assert config.provider is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 configuration files like 'secret' or '.env' might be inadvertently included in the LLM context.

Key Assertions:

- The function `get_default_config()` returns an object with a list of exclude globs.
- Any string containing 'secret' is found in the excludes list.
- Any string containing '.env' is found in the excludes list.
- The excludes list does not contain any strings that start with 'secret' or '.env'.
- The function `get_default_config()` returns a list of globs that are not empty.
- The excludes list contains only non-empty globs.
- Any string containing 'secret' is found in the excludes list after filtering out empty globs.
- Any string containing '.env' is found in the excludes list after filtering out empty globs.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the output of the full pipeline is deterministic, i.e., the nodes are reported in a consistent order.

Why Needed: This test prevents regression where the order of reports changes due to external factors or system updates.

Key Assertions:

- nodeids should be sorted consistently.
- nodeid 'z_test.py::test_z' should come first in the list.
- nodeid 'a_test.py::test_a' should come second in the list.
- nodeid 'm_test.py::test_m' should come third in the list.
- The order of nodes is not affected by external factors or system updates.

COVERAGE

src/pytest_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 5ms 🛡️ 5

AI ASSESSMENT

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

Why Needed: Prevents regression where the test suite is empty, potentially causing invalid reports.

Key Assertions:

- The total count of tests in the report should be zero.
- The summary section of the report should have a total count of zero.
- All other sections of the report should not contain any data.

COVERAGE

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

PASSED `tests/test_integration_gate.py::TestFullPipeline::test_html_report_generation` 30ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the full pipeline generates an HTML report.

Why Needed: This test prevents a potential bug where the HTML report is not generated correctly due to incorrect configuration.

Key Assertions:

- The HTML report should be created at the specified path.
- The content of the HTML report should contain the string '
- The content of the HTML report should contain the string 'test_pass'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	113 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_integration_gate.py::TestFullPipeline::test_json_report_generation` 53ms 🛡️ 7

AI ASSESSMENT

Scenario: Test that the full pipeline generates a valid JSON report.

Why Needed: This test prevents regression where the full pipeline fails to generate a valid JSON report due to incorrect configuration or missing dependencies.

Key Assertions:

- The 'report_json' and 'report_html' paths are created in the test directory.
- A JSON file named 'report.json' is written with the correct schema version, summary statistics, and number of tests.
- The total number of passed tests is 1, failed tests is 1, and skipped tests is 1.
- The 'schema_version' field matches the expected value of SCHEMA_VERSION.
- The 'summary' section contains the correct data: total tests = 3, passed tests = 1, failed tests = 1, skipped tests = 1.

COVERAGE

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

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

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

AI ASSESSMENT

Scenario: Verify that the ReportRoot class has required fields.

Why Needed: This test prevents a potential bug where the report root is missing essential metadata.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that `RunMeta` has the required 'aggregation_fields' key.

Why Needed: Prevents regression where `is_aggregated` is False, potentially causing incorrect aggregation behavior.

Key Assertions:

- The 'is_aggregated' key should be present in the data.
- The 'run_count' key should be present in the data.
- The value of 'aggregation_policy' should only include 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 RunMeta object contains status fields.

Why Needed: This test prevents a potential regression where the RunMeta object is not properly initialized with status fields.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verifies that the schema version is defined and conforms to a semver-like format.

Why Needed: Prevents regression where the schema version is not defined or does not conform to a semver-like format.

Key Assertions:

- The schema version should be present in the test.
- The schema version should be a string that represents a valid semver-like format (e.g., '1.2.3').
- The schema version should contain at least one dot (.) character, indicating it is defined.
- The schema version should not start with a zero (0).
- The schema version should not end with a trailing period (.).
- The schema version should be a valid semver number (e.g., '1.2.3').

COVERAGE

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

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

AI ASSESSMENT

Scenario: The `TestSchemaCompatibility` class is tested to ensure that the `test_case_has_required_fields` method verifies the presence of required fields in a test case.

Why Needed: This test prevents a potential bug where a test case may not have all necessary fields, potentially causing issues with data validation or schema compatibility.

Key Assertions:

- The 'nodeid' field is present in the `data` dictionary.
- The 'outcome' field is present in the `data` dictionary.
- The 'duration' field is present in the `data` dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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 correctly returns a `GeminiProvider` instance when the `provider` parameter is set to `'gemini'`.

Why Needed: This test prevents a potential bug where the `get_provider` function incorrectly returns an incorrect provider type.

Key Assertions:

- The `__class__.__name__` attribute of the returned provider should be `GeminiProvider`.
- The `provider` parameter is set to `'gemini'`.
- The `get_provider` function correctly returns a `GeminiProvider` instance when the `provider` parameter is set to `'gemini'`.
- The `__class__.__name__` attribute of the returned provider matches the expected value (`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 correctly returns an instance of LiteLLMProvider when a specific provider is specified.

Why Needed: This test prevents a potential bug where the correct provider is not returned if an incorrect or unsupported provider is provided.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.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_noop

Why Needed: This test prevents a potential bug where the LLM is not initialized correctly with a 'none' provider.

Key Assertions:

- The `get_provider` function should return an instance of `NoopProvider` when given a configuration with 'provider='none'.
- The `provider` attribute of the returned `NoopProvider` instance should be set to `'none'`.
- The `config` object passed to `get_provider` has a valid `provider` key.
- The `provider` value in the `config` object is correctly converted to a string.
- The `provider` attribute of the resulting `NoopProvider` instance does not contain any non-'none' characters.
- The `provider` attribute of the resulting `NoopProvider` instance has the correct type (str).
- The `provider` attribute of the resulting `NoopProvider` instance is set to a string value without any quotes.
- The `provider` attribute of the resulting `NoopProvider` instance does not contain any non-string characters.

COVERAGE

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

Why Needed: This test prevents a potential bug where an incorrect provider type is returned.

Key Assertions:

- The provider should be of type OllamaProvider.
- The provider's __class__ attribute should match 'OllamaProvider'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that unknown providers are correctly identified and cause a ValueError.

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

Key Assertions:

- The function `get_provider(config)` should be called with a valid provider.
- A ValueError exception should be raised when the provider is unknown.
- The error message should contain the string 'unknown'.
- The error message should be case-insensitive (e.g., 'Unknown Provider' instead of '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 the LlmProvider interface.

Why Needed: Prevents a potential bug where `NoopProvider` is not implementing all required methods of LlmProvider.

Key Assertions:

- Should have required methods: annotate, is_available, get_model_name, config
- The provider should be able to provide the model name from its configuration
- The provider should be able to return a boolean indicating if it's 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/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the annotate method returns an empty LlmAnnotation object when no annotation is provided.

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

Key Assertions:

- annotation is of type LlmAnnotation
- annotation scenario is empty
- annotation why_needed is empty
- annotation key_assertions are empty

COVERAGE

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

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

AI ASSESSMENT

Scenario: The `get_model_name` method of the `NoopProvider` class is called with an empty configuration.

Why Needed: Without this test, a bug or regression may occur where the `get_model_name` method returns an empty string when given an empty configuration.

Key Assertions:

- assert provider.get_model_name() == ''
- assert isinstance(provider.get_model_name(), str)
- assert provider.get_model_name().startswith('')
- assert provider.get_model_name().endswith('')
- assert len(provider.get_model_name()) == 0
- assert provider.get_model_name() != 'noop' # This is not the expected 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/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 66)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Verify that the NoopProvider instance is available.

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

Key Assertions:

- The `is_available()` method should return True for any valid configuration.
- The `is_available()` method should raise an exception if there are any invalid configurations.
- The `is_available()` method should call the underlying provider's `__call__` method without raising any exceptions.
- The `is_available()` method should not throw a `ValueError` exception when called with a valid configuration.
- The `is_available()` method should not raise an exception when called with an invalid configuration.
- The NoopProvider instance should be created successfully for each test case.
- The provider's underlying implementation should be available without any issues.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	5 lines (ranges: 52-53, 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 regression where the annotation summary is not printed, potentially causing confusion or errors in the testing process.

Key Assertions:

- The function `get_provider` from `pytest_llm_report.llm.annotator` is called with a valid configuration.
- The `test_case` nodeid matches the expected scenario.
- The annotation summary is printed when annotations run successfully.

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 LLM annotator progress reporting for multiple tests.

Why Needed: This test prevents regression where the LLM annotation progress is not reported correctly when running multiple tests simultaneously.

Key Assertions:

- The LLM annotation should report progress for each test.
- The LLM annotation should display the correct test ID in its message.
- The LLM annotation should append a string to the messages list indicating the start of annotations for the current test.
- The LLM annotation should not append any additional strings beyond the initial message.
- The progress callback should be called with the correct number of tests annotated.
- The progress callback should call the provider with the correct test ID.
- The provider should have a valid cache directory.
- The test result outcome should be 'passed' for this test.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The 'llm_opt_out' attribute is set to True for tests with this scenario.
- The LLM annotation is only called for tests where 'llm_opt_out' is False.
- No LLM annotations are called for tests where 'llm_opt_out' is True or the maximum number of tests has been reached.
- The provider function returns a FakeProvider instance that calls the correct provider.
- The LLM annotation is not called for test with this scenario and llm_opt_out=False.
- No LLM annotations are called for all tests where llm_opt_out=True or the maximum number of tests has been reached.
- The 'llm_max_tests' attribute is set to 1 in the config, which limits the number of tests to 1.
- The LLM annotation is only called once per test with this scenario and llm_opt_out=False.
- No LLM annotations are called for all tests where llm_opt_out=True or the maximum number of tests has been reached.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where LLM annotations may not respect the requests-per-minute rate limit, leading to inaccurate or delayed results.

Key Assertions:

- The provider's calls should match the expected list of node IDs.
- The sleep function should be called twice with values 2.0 and 4.0 respectively.
- The calls should not exceed the configured requests-per-minute rate limit (30).
- The calls should include all nodes specified in the tests (tests/test_a.py::test_a and tests/test_b.py::test_b).

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: Tests for annotating tests with unavailable providers should be skipped.

Why Needed: This test prevents regression by ensuring that annotation is skipped when a provider is unavailable.

Key Assertions:

- The `is_available` method of the `UnavailableProvider` class returns False.
- The `get_provider` function from `pytest_llm_report.llm.annotator` calls the `is_available` method of the `UnavailableProvider` class with the provided configuration.
- When a provider is unavailable, the `is_available` method should return False.
- The test should fail when an unavailable provider is used to annotate tests.
- The message 'is not available' should be printed when an unavailable provider is used to annotate tests.
- The `annote` function from `pytest_llm_report.llm.annotator` should call the `get_provider` function with a configuration that includes the `UnavailableProvider` class.
- When an unavailable provider is used to annotate tests, the `annote` function should not be able to proceed without skipping the annotation.

COVERAGE

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

Why Needed: This test prevents a regression where annotations are not being used as expected due to caching issues.

Key Assertions:

- The `provider.calls` assertion checks if the provider was called before annotating tests.
- The `test.llm_annotation` assertion checks if the annotation is set correctly and matches the scenario.
- The `provider_next.annotate` assertion checks if the annotation is not called when it should be.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the `test_required_fields` function checks for the presence of 'scenario' and 'why_needed' keys in the ANNOTATION_JSON_SCHEMA.

Why Needed: This test prevents a potential bug where the schema is not correctly validated if the required fields are missing or empty.

Key Assertions:

- The 'scenario' key should be present in the ANNOTATION_JSON_SCHEMA.
- The 'why_needed' key should also be present in the ANNOTATION_JSON_SCHEMA.
- If 'scenario' and 'why_needed' keys are not present, the test will fail with an error message indicating a required field 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/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.from_dict method correctly parses a dictionary into an annotation.

Why Needed: This test prevents potential bugs or regressions in the AnnotationSchema class where it may not be able to parse dictionaries correctly.

Key Assertions:

- checks password
- checks username
- schema.scenario matches 'Tests user login'
- schema.why_needed matches 'Prevents auth bypass'
- len(schema.key_assertions) is equal to 2
- schema.confidence is greater than or equal to 0.95

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the AnnotationSchema handles an empty input by creating a schema from an empty dictionary.

Why Needed: This test prevents regression where the AnnotationSchema is not correctly handling empty inputs, potentially leading to incorrect validation or errors.

Key Assertions:

- schema.scenario = "" (empty string)
- schema.why_needed = "" (empty string)""
- assert schema.scenario == "" (checks if the assertion matches an expected value)
- assert schema.why_needed == "" (checks if the assertion matches an expected value)
- assert isinstance(schema, AnnotationSchema) (checks if the assertion is a valid instance of AnnotationSchema)
- schema.from_dict({}) should raise a ValueError or return None (checks if the from_dict method raises an exception or returns None)

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the AnnotationSchema.from_dict method correctly handles a partial input scenario.

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

Key Assertions:

- The schema's 'scenario' property is set to 'Partial only'.
- The schema's 'why_needed' property is empty, indicating that no specific bug or regression was prevented by this test.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the schema is not properly defined with required fields, potentially leading to incorrect validation of contract data.

Key Assertions:

- The 'scenario' field should be present in the schema's properties.
- The 'why_needed' field should also be present in the schema's properties.
- The 'key_assertions' field should contain assertions about the required fields within the schema.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 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: tests/test_llm_contract.py::TestAnnotationSchema::test_schema_to_dict

Why Needed: Prevents regression of bug Y when schema is serialized to dict.

Key Assertions:

- data['scenario'] == 'Tests feature X'
- data['why_needed'] == 'Prevents bug Y'
- data['key_assertions'] in data

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `NoopProvider` is returned when a factory configuration with 'provider' set to 'none' is used.

Why Needed: This test prevents a potential regression where the `NoopProvider` is not returned for provider='none'.

Key Assertions:

- The function `get_provider(config)` returns an instance of `NoopProvider` when the configuration has a 'provider' set to 'none'.
- The `NoopProvider` instance is correctly created and assigned to the variable `provider`.
- The `isinstance(provider, NoopProvider)` assertion passes, indicating that the correct class is returned.
- The test does not fail when using a factory configuration with 'provider' set to 'none'.
- The `get_provider(config)` function handles cases where the provider is not specified correctly.
- No exceptions are raised during the execution of this test.
- The test covers all possible scenarios for the given scenario.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `NoopProvider` class is mistakenly implemented as an LLM provider instead of a no-op.

Key Assertions:

- The `provider` variable should be an instance of `LlmProvider`.
- The `provider` variable should not have any attributes or methods other than those inherited from `LlmProvider`.
- The `provider` variable should not inherit any attributes or methods from the `NoopProvider` class.
- The `provider` variable's type should be correctly set to `LlmProvider` using the `isinstance()` function.
- Any additional attributes or methods in the `provider` object should be removed after inheritance.
- The `provider` object should not have any unexpected behavior when accessed through its 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`	`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 function should return an empty annotation when the provided test case does not match any known node IDs.

Why Needed: This test prevents a regression where the NoopProvider returns incorrect annotations for tests that do not match any known node IDs.

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: Verify that the `annotate` method of `NoopProvider` returns an LlmAnnotation-like object with the correct attributes.

Why Needed: This test prevents a potential regression where the `annotate` method does not return the expected annotation.

Key Assertions:

- The result has the attribute 'scenario' and it is set to 'Annotate returns LlmAnnotation-like object.'
- The result has the attribute 'why_needed' and it is set to 'This test prevents a potential regression where the `annotate` method does not return the expected annotation.'
- The result has the attribute 'key_assertions' and it contains the expected checks

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- The provider should return a non-empty result for an empty code.
- The provider should not raise any errors or exceptions when handling empty code.
- The provider should be able to correctly annotate the test with a valid outcome.
- The annotation should include the nodeid and outcome of the test.
- The configuration should not affect the behavior of the provider.
- The result should not be None, indicating that the contract handled the empty code successfully.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the provider might throw an error when handling a `None` context.

Key Assertions:

- The `provider.annotate()` method should return `None` instead of raising an exception.
- The `provider.annotate()` method should not raise an exception if the input `test` is `None`.
- The `provider.annotate()` method should correctly handle the case where the input `test` is `None` and returns a valid `TestCaseResult` object.
- The `provider.annotate()` method should not throw any exceptions when handling a `None` context.
- The `provider.annotate()` method should preserve the original value of the `nodeid` attribute in the `TestCaseResult` object.
- The `provider.annotate()` method should preserve the original value of the `outcome` attribute in the `TestCaseResult` object.
- The `provider.annotate()` method should not modify the original values of these attributes in the `TestCaseResult` object.

COVERAGE

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

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

AI ASSESSMENT

Scenario: All providers should have an annotate method.

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

Key Assertions:

- provider_name in ['none', 'ollama', 'litellm', 'gemini']
- hasattr(provider, 'annotate')
- callable(provider.annotate)
- provider.annotate() should return a callable object

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	15 lines (ranges: 52-53, 245, 247, 249-250, 252-253, 255, 257-258, 260, 262-263, 265)
src/pytest_llm_report/llm/gemini.py	7 lines (ranges: 134, 136-139, 141-142)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The `annotate` method of the `GeminiProvider` class is being tested when it handles a context that is too large.

Why Needed: This test prevents a potential bug where the `annotate` method fails with an exception due to an excessive memory usage in cases with very large contexts.

Key Assertions:

- The `context_size` attribute of the `GeminiProvider` instance should be less than or equal to 1000.
- The `annotate` method should not raise a `MemoryError` when called with a context that exceeds 1000 bytes in size.
- The `context_size` attribute of the `GeminiProvider` instance should be updated correctly after calling the `annotate` 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/llm/base.py	9 lines (ranges: 52-53, 72, 75-76, 80, 165, 167, 175)
src/pytest_llm_report/llm/gemini.py	155 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-221, 233, 245-248, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 346, 348-350, 352-355, 360-363, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417-418, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the provider incorrectly reports an installation issue without providing any useful information about the actual problem.

Key Assertions:

- assert annotation.error == 'litellm not installed. Install with: pip install litellm'
- provider.annotate(test, 'def test_case(): assert True')
- test.test_case() should raise a mock_import_error('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 GeminiProvider annotates missing tokens correctly.

Why Needed: To prevent a TypeError when trying to annotate a test with an undefined API token.

Key Assertions:

- The annotation should raise a ValueError indicating that GEMINI_API_TOKEN is not set.
- The annotation should include a message explaining the expected behavior (i.e., 'GEMINI_API_TOKEN is not set')
- The annotation should provide a clear indication of what needs to be set (API token) before annotating a test
- The annotation should return an error code indicating that GEMINI_API_TOKEN is missing
- The annotation should include the actual API token value if it exists

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify Gemini provider annotates records tokens correctly.

Why Needed: Prevents regression in token usage logging.

Key Assertions:

- The 'annotate' method of the GeminiProvider is called with a test function that checks for token usage.
- The 'annotate' method logs usage metadata, including the total number of tokens recorded.
- The 'annotate' method verifies if the limiter has at least one record of token usage.
- The limiter's token usage is verified to be 1 token with a count of 123.
- The rate limits logic is tested by verifying it ran without error.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	183 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-101, 103, 105, 107-109, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-223, 225-227, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-343, 346, 348-350, 352-355, 360-366, 368, 370-371, 374-377, 381-382, 385-387, 391-392, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)

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

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

AI ASSESSMENT

Scenario: The `annotate_retries_on_rate_limit` method of the `GeminiProvider` class should retry annotating requests when rate limiting is exceeded.

Why Needed: This test prevents a potential issue where the `annotate_retries_on_rate_limit` method does not retry after exceeding the rate limit, potentially causing the service to become unresponsive or fail with an error.

Key Assertions:

- The `rate_limit` attribute of the `GeminiProvider` instance is set correctly before calling `annotate_retries_on_rate_limit`.
- The `annotate_retries_on_rate_limit` method retries annotating requests when rate limiting is exceeded. The retry attempts are limited to a reasonable number of times.
- The `rate_limit` attribute is reset after each retry attempt, allowing the service to recover from rate limiting issues. If `rate_limit` is not reset, it may lead to infinite retries and potential service degradation.
- The `annotate_retries_on_rate_limit` method does not raise an exception when rate limiting is exceeded, preventing the test from failing due to expected behavior.
- The `annotate_retries_on_rate_limit` method uses a reasonable number of retry attempts (e.g., 3-5) before giving up and returning without annotating the request. This helps prevent overwhelming the service with retries and potential denial-of-service attacks.
- The `rate_limit` attribute is not reset after each retry attempt, allowing the service to recover from rate limiting issues. If `rate_limit` is not reset, it may lead to infinite retries and potential service degradation.
- The `annotate_retries_on_rate_limit` method does not log any relevant information when rate limiting is exceeded, preventing the test from detecting potential issues or errors in the service.

COVERAGE

src/pytest_llm_report/collector.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 a daily limit when used with the `rotate_models_on_daily_limit` fixture.

Why Needed: This test prevents a potential bug where the model rotation is not applied correctly due to an incorrect implementation of the `rotate_models_on_daily_limit` fixture.

Key Assertions:

- The `annotate` method should rotate models on the daily limit.
- The `rotate_models_on_daily_limit` fixture should be able to rotate models without any issues.
- The model rotation is applied correctly and does not exceed the daily limit.

COVERAGE

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

Why Needed: This test prevents a regression where the annotate function does not skip annotations due to the daily limit being reached.

Key Assertions:

- The annotate function should skip any annotations that would exceed the daily limit.
- Any annotations that are created after the daily limit has been exceeded should be skipped.
- The annotate function should not attempt to create new annotations when the daily limit is reached.
- The annotation count should decrease by 1 when the daily limit is exceeded.
- The total number of annotations created should be less than or equal to the daily limit after exceeding it.
- The annotate function should throw an exception when the daily limit is exceeded and no new annotations are allowed.
- Any exceptions thrown during annotation creation should not propagate up the call stack.

COVERAGE

src/pytest_llm_report/collector.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 successful responses correctly.

Why Needed: Prevents regressions caused by incorrect annotation of failed responses.

Key Assertions:

- The annotated response contains the expected scenario and why-need information.
- The annotated response includes the correct key assertions.
- The confidence level is set to a reasonable value (0.8 in this case).
- The captured model is correctly identified as 'gpt-4o'.
- The system role is correctly associated with the message 'def test_login()'.
- The tests/test_auth.py::test_login message is present in the response.
- The successful login scenario is included in the response content.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/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 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 or near-normal values after 24 hours.
- The model's inference time should decrease significantly after 24 hours.
- The model's memory usage should decrease significantly after 24 hours.
- The model's warnings and errors should be cleared after 24 hours.
- The model's training data should not have been exhausted within the last 24 hours (if applicable).
- The model's inference requests should be able to complete successfully without any timeouts or exceptions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	190 lines (ranges: 39-42, 45-46, 48-50, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-188, 190-191, 193-194, 196, 200-208, 210-211, 213-214, 217-222, 225-227, 252-254, 274-277, 280-283, 286-290, 292-295, 297-298, 300-301, 315, 317-320, 322-325, 327-328, 330, 335-341, 343, 346, 348-350, 352-355, 360-366, 368, 370, 372-377, 381-382, 385-387, 391-393, 396-399, 401-402, 405, 408-410, 412-414, 417, 419, 421-424, 428, 430-434, 437-440, 442-443, 445-447)

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

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

AI ASSESSMENT

Scenario: The 'fetch_available_models' method of the Gemini provider returns an error when no models are available.

Why Needed: This test prevents a regression where the 'fetch_available_models' method returns an error instead of raising a meaningful exception.

Key Assertions:

- assertRaisesRegex with 'GeminiError' and 'No models available.'
- assertRaisesRegex with 'GeminiError' and 'No models found.'
- assertRaisesRegex with 'GeminiError' and 'Model not found in database.'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	65 lines (ranges: 134, 136-139, 141-142, 280, 282-283, 286-290, 292-295, 297-298, 300-301, 346, 348-350, 352-355, 360-363, 374-377, 385, 387, 391-392, 396-402, 405, 408-410, 412-414, 417-418, 428, 430-432, 435-436)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The model list should refresh after a specified interval.

Why Needed: This test prevents regression when the model is not refreshed immediately after an interval.

Key Assertions:

- model_list is updated with new models after the specified interval.
- no exception is raised if no models are available to update.
- models are only added to the list if they have been trained within the specified interval.
- the model list size does not exceed a certain threshold.
- the test can be run in parallel without affecting each other's results.
- the refresh interval can be adjusted dynamically based on system load.

COVERAGE

`src/pytest_llm_report/collector.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 in the annotation.

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

Key Assertions:

- assert 'boom' in annotation.error
- annotation.error is an instance of RuntimeError
- annotation.error contains the string 'boom'
- annotation.error is raised by fake_completion()
- fake_completion raises a RuntimeError
- fake_completion() does not raise any other exception

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175)
src/pytest_llm_report/llm/litellm_provider.py	22 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 78-79, 81-82, 84-85)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that LiteLLMProvider rejects invalid key_assertions payloads.

Why Needed: To prevent the test from passing when an invalid key_assertions payload is provided.

Key Assertions:

- The 'key_assertions' parameter must be a list.
- Invalid response: key_assertions must be a list
- Key assertion error message should include the expected format.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The LiteLLMProvider annotates a missing dependency in the provided test case.

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

Key Assertions:

- annotation.error == 'litellm not installed. Install with: pip install litellm'
- provider.annotate(test, "def test_case(): assert True")
- test.test_case() is False

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents regression due to fake completion of LiteLLMProvider.

Key Assertions:

- The annotation contains the correct scenario.
- The annotation contains the correct why needed message.
- The annotation contains the correct key assertions.
- The annotation has a confidence level of 0.8.
- The captured model is 'gpt-4o'.
- The captured messages contain the expected system role and function calls.
- The captured messages contain the expected test login function call.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the LiteLLM provider detects installed modules.

Why Needed: Prevents a potential bug where the provider does not detect installed modules.

Key Assertions:

- The `is_available()` method of the `LiteLLMProvider` class should return True when the 'litellm' module is available in the system's modules.
- The `is_available()` method should raise an error if the 'litellm' module is not installed or not found in the system's modules.
- The provider should correctly detect the presence of the 'litellm' module even if it is not a standard Python package.
- The provider should handle cases where the 'litellm' module is installed but not imported as a module (e.g., as a package)
- The provider should raise an error when trying to import the 'litellm' module as a module, indicating that it is not available
- The provider should correctly handle cases where the 'litellm' module has been removed from the system's modules

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 107, 110-111)
src/pytest_llm_report/llm/litellm_provider.py	3 lines (ranges: 94-95, 97)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_llm_providers.py::TestOllamaProvider::test_annotate_fallbacks_on_context_length_error 1ms 🛡️ 6

AI ASSESSMENT

Scenario: The test verifies that the annotate method handles context length errors correctly.

Why Needed: This test prevents a potential regression where the annotate method fails to handle context length errors.

Key Assertions:

- If the input `context` is longer than the maximum allowed length, the function should return an error message.
- The function should raise a ValueError with a meaningful error message when the input `context` is too long.
- The function should not silently ignore the input `context` and instead raise an exception.
- The function should provide a clear and descriptive error message that explains why the context length was exceeded.
- The function should handle cases where the input `context` is an empty string or None.
- The function should return an error message with a specific format (e.g., 'Context too long: ...')
- The function should raise an exception with a specific error code (e.g., 'EXC_CONTEXT_LENGTH_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: Test OllamaProvider::test_annotate_handles_call_error verifies that the annotation of a call error to Ollama provider prevents regression.

Why Needed: This test prevents regression in case of call errors to Ollama provider, ensuring the correctness of annotations.

Key Assertions:

- The annotation should indicate that the call was not successful and returned an error.
- The error message should be 'Failed after 3 retries. Last error: boom'.
- The test should pass even if the system prompt is different from the Ollama provider's system prompt.
- The test should fail with a non-zero exit code (e.g., 1) when the call to Ollama provider fails.
- The annotation should not be affected by the number of retries.
- The annotation should indicate that the call was not successful and returned an error even if the system prompt is different from the Ollama provider's system prompt.
- The test should fail with a non-zero exit code (e.g., 1) when the call to Ollama provider fails, regardless of the number of retries.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175)
src/pytest_llm_report/llm/ollama.py	16 lines (ranges: 40-41, 47, 50, 52, 54-55, 57-59, 71-72, 74-75, 77-78)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The Ollama provider should report an error when annotating a function that uses the missing httpx dependency.

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

Key Assertions:

- The annotation message should include the correct error message for installing httpx.
- The annotation message should not be empty.
- The annotation message should contain the exact string 'httpx not installed.'
- The annotation message should not contain any other relevant information that could lead to incorrect error messages.
- The annotation message should not be too long and only include the necessary information for installation.
- The annotation message should not contain any typos or grammatical errors.
- The annotation message should provide a clear and concise explanation of what needs to be installed.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents authentication-related bugs in the annotator.

Key Assertions:

- Verify status code and validate token response
- Check if the response contains a valid JSON object
- Assert that the 'response' key is present in the JSON object
- Verify that the 'why_needed' field matches the expected reason for failure
- Validate that the 'key_assertions' list includes all necessary checks

COVERAGE

src/pytest_llm_report/collector.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: The Ollama provider makes a successful API call to generate text.

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

Key Assertions:

- The `provider._call_ollama` method returns the expected response from the Ollama model.
- The `captured.json['model']` attribute matches the provided `model` in the configuration.
- The `captured.json['prompt']` attribute matches the provided `prompt` in the configuration.
- The `captured.json['system']` attribute matches the provided `system` in the configuration.
- The `captured.json['stream']` attribute is set to `False` as expected.
- The `timeout` attribute is set to 60 seconds as expected.

COVERAGE

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

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

Key Assertions:

- The captured response from the API should contain the default model.
- The captured response from the API should be 'ok'.
- The captured response from the API should have a 'model' key with value 'llama3.2'.

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/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	16 lines (ranges: 114, 116-123, 127-130, 132, 134-135)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the Ollama provider returns False when the server is unavailable.

Why Needed: This test prevents a regression where the provider fails to return an error when the server is not running.

Key Assertions:

- The function _check_availability() of the OllamaProvider instance should raise a ConnectionError exception.
- The function _check_availability() of the OllamaProvider instance should return False.
- The function _check_availability() of the OllamaProvider instance should not have any other return value.
- The function _check_availability() of the OllamaProvider instance should not raise a TypeError exception.
- The function _check_availability() of the OllamaProvider instance should not be able to return True.
- The function _check_availability() of the OllamaProvider instance should not have any side effects.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: To prevent a regression where the provider incorrectly reports availability when it's not available (status code 500).

Key Assertions:

- assert provider._check_availability() is False
- assert FakeResponse().status_code == 500
- assert config.provider != 'ollama'
- assert isinstance(provider, OllamaProvider)
- assert isinstance(config, Config)
- assert isinstance(fake_httpx, SimpleNamespace)
- assert fake_get.__name__ == 'fake_get'
- assert fake_get.__doc__ is None

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the provider fails to check availability when it's not available.

Key Assertions:

- The '/api/tags' URL is present in the provided host.
- The response status code is 200 (OK) for the '/api/tags' endpoint.
- The 'ollama_host' configuration parameter is set correctly to 'http://localhost:11434'.
- The provider's `_check_availability()` method returns True.

COVERAGE

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

Why Needed: This test prevents a potential bug where the provider might be incorrectly identified as non-local.

Key Assertions:

- provider is an instance of OllamaProvider
- is_local() returns True for the provided 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/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 having an error.

Key Assertions:

- The `annotation.error` attribute is set to 'Failed to parse LLM response as JSON'.
- The `provider._parse_response('not-json')` method returns an instance of `OllamaProviderError`.
- The `annotation.error` attribute contains the string 'Failed to parse LLM response as JSON'.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- - The 'key_assertions' field must be a list.
- - Invalid values are not allowed in this field.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The provided test verifies that the Ollama provider correctly extracts JSON from markdown code fences.

Why Needed: This test prevents a potential bug where the provider fails to parse JSON in code fences, potentially leading to incorrect or incomplete annotations.

Key Assertions:

- The response is not empty.
- The response contains valid JSON syntax.
- The response does not contain any non-JSON characters (e.g. whitespace, special characters).
- The response does not contain any invalid JSON syntax (e.g. missing or mismatched brackets, quotes).
- The response is a valid JSON object (i.e. an object with a 'text' property and optional 'metadata' properties).
- The provider correctly handles nested objects and arrays within the JSON.
- The provider correctly handles quoted strings within the JSON.
- The provider does not attempt to parse any invalid or malformed JSON.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 fence without any language specification.

Why Needed: This test prevents a potential bug where the provider fails to extract JSON from plain fences with no specified language.

Key Assertions:

- ...
- ...
- ...
- The extracted JSON should be valid JSON syntax without any extra characters or whitespace.
- The response should not contain any language-specific keywords or phrases.
- ...
- ...
- The provider's error message should indicate that no language was specified for the fence.

COVERAGE

src/pytest_llm_report/collector.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 and verifies correct configuration.

Why Needed: Prevents bugs in the Ollama provider by ensuring it correctly configures itself with a valid response.

Key Assertions:

- assert annotation.scenario == 'Tests feature'
- assert annotation.why_needed == 'Stops bugs'
- assert annotation.key_assertions == ['assert a', 'assert b']
- 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` correctly serializes to a dictionary.

Why Needed: This test prevents a bug where the serialization of `CoverageEntry` is incorrect.

Key Assertions:

- The 'file_path' key in the serialized dictionary should be exactly 'src/foo.py'.
- The 'line_ranges' key in the serialized dictionary should match the provided string.
- The 'line_count' key in the serialized dictionary should be equal to the expected value of 10.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	4 lines (ranges: 254-257)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_models.py::TestCollectionError::test_to_dict 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `to_dict()` method of CoverageEntry to ensure it correctly serializes a coverage entry.

Why Needed: This test prevents regressions where the `to_dict()` method fails to serialize a coverage entry with invalid or missing data.

Key Assertions:

- The `file_path` key in the serialized dictionary should match the original value.
- The `line_ranges` key in the serialized dictionary should match the original value.
- The `line_count` key in the serialized dictionary should match the original value.
- Any missing keys (e.g. `coverage_type`, `start_line`, etc.) should be ignored or raise an error.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	3 lines (ranges: 207-209)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test CoverageEntry to_dict serialization correctness.

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

Key Assertions:

- The 'file_path' key should match the expected value.
- The 'line_ranges' key should contain the correct ranges and values.
- The 'line_count' key should match the expected value.
- A KeyError should be raised if the 'file_path', 'line_ranges', or 'line_count' keys are missing 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	4 lines (ranges: 40-43)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

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

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

Key Assertions:

- annotation.scenario == "" (empty string)
- annotation.why_needed == "Empty annotation should have default values." (description of why it needs to be tested)
- annotation.key_assertions == [] (expected empty list for key assertions)
- assert annotation.confidence is None (expected confidence to be None for an empty annotation)
- assert annotation.error is None (expected error to be None for an empty annotation)

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/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 is properly serialized without any optional or missing fields.

Key Assertions:

- The 'scenario' field should be present in the dictionary.
- The 'why_needed' field should also be present in the dictionary.
- The 'key_assertions' field should not include the 'confidence' field, as it is an optional 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	8 lines (ranges: 104-107, 109, 111, 113, 115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test to dictionary with all fields

Why Needed: Prevents data loss due to missing fields in the output.

Key Assertions:

- Asserts that the 'scenario' field is present and matches the expected value.
- Asserts that the 'confidence' field has a value greater than or equal to 0.95.
- Asserts that the 'context_summary' field contains the correct mode ('minimal') and bytes count (1000).

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test default report schema version and empty lists.

Why Needed: Prevents regression by ensuring the default report has a valid schema version and no collection errors or warnings.

Key Assertions:

- The 'schema_version' key in the report dictionary should be equal to SCHEMA_VERSION.
- The 'tests' key in the report dictionary should be an empty list.
- The 'warnings' key in the report dictionary should not exist (i.e., its value should be None).
- The 'collection_errors' key in the report dictionary should not exist (i.e., its value should be None).

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test Report with Collection Errors should include them.

Why Needed: This test prevents a regression where the report does not include collection errors.

Key Assertions:

- The 'collection_errors' key in the report dictionary should be present and contain exactly one error.
- The 'nodeid' value of the first error in the 'collection_errors' list should match the provided node id.
- All other values in the 'collection_errors' list (if any) should have a valid 'message' property.

COVERAGE

`src/pytest_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 the presence of warnings in a ReportRoot instance.

Why Needed: This test prevents a regression where warnings are not reported when there is no coverage.

Key Assertions:

- The length of the 'warnings' list should be exactly 1.
- The first element of the 'warnings' list should have code 'W001'.

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the output of `ReportRoot` is sorted by nodeid.

Why Needed: This test prevents a regression where tests are not sorted correctly by nodeid, potentially causing incorrect report generation.

Key Assertions:

- The list of nodeids in the output matches the expected order.
- Each nodeid appears exactly once in the list.
- All nodeids are present in the input data.
- Nodeids are in ascending order.
- No duplicate nodeids are present in the output.
- Nodeids are not empty.
- The test passes if all assertions pass.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test `test_to_dict_with_detail` verifies that the `to_dict()` method of a `ReportWarning` object returns a dictionary with the correct detail.

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

Key Assertions:

- The 'detail' key should be present in the returned dictionary.
- The value of the 'detail' key should match '/path/to/file'.
- The 'message' key is not included in the returned dictionary.

COVERAGE

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

PASSED `tests/test_models.py::TestReportWarning::test_to_dict_without_detail` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'test_to_dict_without_detail' verifies that a ReportWarning object is created without detailed warnings.

Why Needed: This test prevents the creation of unnecessary detailed warnings when no coverage information is available.

Key Assertions:

- The warning object should be created with only the required code and message keys.
- The 'detail' key should not be present in the warning object.
- The 'message' key should contain the expected warning message.
- The 'code' key should match the expected warning code.
- The 'detail' value should be an empty string or None to indicate no coverage information is 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/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: Prevent regression where RunMeta is missing aggregation fields, potentially leading to incorrect aggregation results.

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

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test LLM traceability fields are included when enabled.

Why Needed: This test prevents regression by ensuring that the LLM traceability fields are properly set to true when llm_annotations_enabled is True.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	40 lines (ranges: 364-380, 382, 385, 387, 390, 393, 395, 397, 399, 401, 403, 407-419)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the `non_aggregated_excludes_source_reports` method of `RunMeta` does not include source reports.

Why Needed: This test prevents a regression where non-aggregated report results do contain source reports.

Key Assertions:

- source_reports is not included in the report dictionary
- is_aggregated is set to False for this meta

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test RunMeta to dict with all optional fields.

Why Needed: Prevents regression in case of missing or outdated metadata.

Key Assertions:

- The 'git_sha' field should be present and have the expected value.
- The 'git_dirty' field should be True.
- The 'repo_version', 'repo_git_sha', and 'plugin_git_sha' fields should be present and have the expected values.
- The 'config_hash' field should be present and have the expected value.
- The length of the 'source_reports' list should be 1.
- All source reports should be dictionaries with the required keys (path, sha256, run_id).
- The 'pytest_invocation', 'pytest_config_summary', and 'run_id' fields should be present and have the expected values.
- The 'is_aggregated' field should be True.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/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: TestRunMeta::test_run_status_fields verifies that RunMeta includes the necessary run status fields.

Why Needed: This test prevents a potential bug where RunMeta is missing required run status fields, potentially leading to incorrect or incomplete results.

Key Assertions:

- The 'exit_code' field should be set to 1.
- The 'interrupted' field should be True.
- The 'collect_only' field should be True.
- The 'collected_count' field should equal 10.
- The 'selected_count' field should equal 8.
- The 'deselected_count' field should equal 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: Verify that the schema version is correctly formatted (semver),

Why Needed: Prevents a potential bug where an incorrect or malformed semver format is used.

Key Assertions:

- The schema version should be split into three parts (e.g., '1.2.3').
- Each part of the schema version should consist only of digits (0-9).
- All parts of the schema version should be non-empty and not empty strings.
- If a part is an empty string, it should be ignored or handled appropriately.
- The first part of the schema version should be greater than 0.
- The second part of the schema version should be less than or equal to 99.
- The third part of the schema version should be less than or equal to 999.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 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: Tests the `ReportRoot` class to ensure it includes the schema version in its report root.

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

Key Assertions:

- The `schema_version` attribute of the `ReportRoot` object should be equal to `SCHEMA_VERSION`.
- The `to_dict()` method of the `ReportRoot` object should return a dictionary with a key named `schema_version` and a value equal to `SCHEMA_VERSION`.
- The `schema_version` property of the `ReportRoot` class should have a value equal to `SCHEMA_VERSION`.
- The `schema_version` attribute of the `ReportRoot` object should be present in its `to_dict()` method.
- The `schema_version` property of the `ReportRoot` class should not be `None` or an empty string.

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the coverage entry data is not properly serialized, potentially leading to incorrect or incomplete coverage reports.

Key Assertions:

- The 'file_path' key in the serialized dictionary matches the expected value.
- The 'line_ranges' key in the serialized dictionary matches the expected value.
- The 'line_count' key in the serialized dictionary matches the expected value.
- All line ranges are correctly formatted (e.g. '1-3', '5, 10-15')
- No missing or extra line ranges are present in the output
- The coverage entry data is not empty

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`8 lines (ranges: 71-78)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

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

AI ASSESSMENT

Scenario: The test verifies that the `to_dict` method of `LlmAnnotation` returns a dictionary with required fields.

Why Needed: This test prevents a regression where the minimal annotation is not properly serialized without the 'confidence' field.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: The test verifies that the `SourceReport` object's `to_dict()` method returns the `run_id` attribute correctly.

Why Needed: This test prevents a potential bug where the `run_id` is not included in the dictionary representation of the `SourceReport` object.

Key Assertions:

- The `run_id` attribute should be present and equal to 'run-1' when the `to_dict()` method is called on a `SourceReport` object with a `run_id` parameter.
- The `run_id` attribute should not be missing or incorrect when the `to_dict()` method is called on a `SourceReport` object without a `run_id` parameter.
- The `run_id` attribute should be present and equal to 'run-1' when the `to_dict()` method is called on a `SourceReport` object with an invalid or missing `run_id` 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	6 lines (ranges: 277-279, 281-283)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the `CoverageEntry` class correctly serializes a coverage report.

Why Needed: This test prevents regression where the coverage report is not properly serialized.

Key Assertions:

- The 'file_path' key in the coverage report should match the provided file path.
- The 'line_ranges' key in the coverage report should contain the expected line ranges.
- The 'line_count' key in the coverage report should match the specified line count.

COVERAGE

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

Why Needed: This test prevents regression that might occur when creating a minimal result with an empty nodeid or outcome.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: tests/test_models.py::TestTestCaseResult::test_result_with_coverage verifies that the test result includes a coverage list.

Why Needed: This test prevents regression by ensuring that the test result accurately reflects the code's coverage.

Key Assertions:

- The 'coverage' key in the result dictionary should contain exactly one entry.
- The 'file_path' value of the first 'coverage' entry should be 'src/foo.py'.
- All lines in the 'coverage' list should have a line count greater than zero.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	22 lines (ranges: 40-43, 161-165, 167, 169, 171, 173, 176-178, 180, 182, 184, 186, 188, 190)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that the `result` object includes a flag indicating LLM opt-out.

Why Needed: Prevents regression where the LLM is enabled by default and the test passes without it.

Key Assertions:

- The value of `llm_opt_out` in the `result` dictionary should be `True`.
- The key `'llm_opt_out'` exists in the `d` dictionary.
- The value of `llm_opt_out` is a boolean value (`True` or `False`).
- The test passes without LLM opt-out enabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	18 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180-182, 184, 186, 188, 190)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test 'test_result_with_rerun' verifies that the TestCaseResult object includes rerun fields.

Why Needed: This test prevents regression by ensuring that reruns are included in the result.

Key Assertions:

- The value of `rerun_count` is 2.
- The value of `final_outcome` is 'passed'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 `test_result_without_rerun_excludes_fields` verifies that the `result` dictionary does not contain 'rerun_count' and 'final_outcome' keys.

Why Needed: This test prevents a regression where the result of a test is rerun, potentially hiding important information about its outcome.

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 'rerun_count' and 'final_outcome' keys should not be present in the `result` 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_options.py::TestConfig::test_default_values` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that default values are set correctly for the Config class.

Why Needed: This test prevents a potential bug where the default configuration values are not set correctly, potentially leading to unexpected behavior or errors in the application.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that `get_default_config()` returns a default configuration with no provider.

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

Key Assertions:

- The function `get_default_config()` should return an instance of `Config`.
- The attribute `provider` on the returned object should be set to `'none'`.
- The value of `provider` should match the expected default provider ('none').

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that the `is_llm_enabled` check returns False for a provider without an LLM.

Why Needed: Prevents regression in case the LLM is not enabled by default.

Key Assertions:

- assert Config(provider='none').is_llm_enabled() is False
- assert Config(provider='ollama').is_llm_enabled() is True

COVERAGE

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

Scenario: Testing the validation of an invalid context mode.

Why Needed: Prevents a potential bug where an invalid context mode is not properly validated and causes unexpected behavior.

Key Assertions:

- The configuration object has been created with an invalid llm_context_mode.
- An error message indicating 'Invalid llm_context_mode' was found in the validation result.
- The error message includes the specific value 'mega_max'.
- A single error is expected as a result of this validation.
- The error message does not contain any additional context or details.
- The test verifies that an error is raised when an invalid context mode is provided.
- The test verifies that the error message contains the correct information about the invalid context mode.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify the test validates for an invalid include phase.

Why Needed: Prevents a potential bug where the validation of invalid include phases is not properly handled.

Key Assertions:

- The function `validate()` returns exactly one error message.
- The error message contains the string 'Invalid include_phase 'lunch_break'.
- The error message includes the specified include phase value.

COVERAGE

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

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

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_options.py::TestConfig::test_validate_numeric_ranges 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test validation of numeric constraints for TestConfig.

Why Needed: Prevents regression when setting invalid numeric ranges for LLM.

Key Assertions:

- The 'llm_context_bytes' value should be at least 1000.
- The 'llm_max_tests' value should be 0 (no limit) or positive.
- The 'llm_requests_per_minute' value should be at least 1.
- The 'llm_timeout_seconds' value should be at least 1.
- The 'llm_max_retries' value should be 0 or positive.
- All numeric constraints must be validated successfully.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Valid configuration is validated successfully without any errors.

Why Needed: This test prevents potential bugs where an invalid or malformed configuration could cause the application to crash or behave unexpectedly.

Key Assertions:

- The `validate()` method of the `Config` object returns an empty list of errors when a valid configuration is provided.
- No error messages are printed to the console indicating that no errors were found in the configuration.
- The test does not fail if the configuration contains invalid or missing values, as it only checks for the absence of errors.
- The `validate()` method correctly identifies and returns an empty list when a valid configuration is provided.
- The function signature and docstring indicate that it should return an empty list in this case.
- The test does not fail if the configuration contains invalid or missing values, as it only checks for the absence of errors.
- The `validate()` method correctly identifies and returns an empty list when a valid configuration is provided.
- The function signature and docstring indicate that it should return an empty list in this case.
- The test does not fail if the configuration contains invalid or missing values, as it only checks for the absence 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	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 loading aggregation options for the `load_config` function.

Why Needed: This test prevents a potential bug where the aggregate policy is set to 'merge' instead of 'merge_run_id'.

Key Assertions:

- The value of `aggregate_dir` should be 'aggr_dir'.
- The value of `aggregate_policy` should be 'merge'.
- The value of `aggregate_run_id` should be 'run-123'.
- The value of `aggregate_group_id` should be '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 handling of invalid integer values in INI files.

Why Needed: Prevents a potential bug where the test crashes due to an invalid integer value in the INI file.

Key Assertions:

- The function `load_config` should not crash when it encounters an invalid integer value in the INI file.
- The default value of `llm_max_retries` is correctly set to 3.
- The test does not verify that the fallback value is incorrect (i.e., `llm_report_max_retries` equals 'garbage').

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	28 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263-267, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: The test verifies that the `llm_coverage_source` option is set to 'cov_dir' after loading the configuration.

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

Key Assertions:

- mock_pytest_config.option.llm_coverage_source
- cfg.llm_coverage_source == 'cov_dir'
- cfg.llm_coverage_source is not None
- cfg.llm_coverage_source is not empty
- cfg.llm_coverage_source does not contain 'default'
- cfg.llm_coverage_source contains 'cov_dir'

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the configuration defaults to 'none' without any explicit option being set, potentially leading to unexpected behavior or errors in subsequent tests.

Key Assertions:

- cfg.provider == 'none'
- cfg.report_html is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	24 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test that CLI options override ini options.

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

Key Assertions:

- ini_value is set to 'cli_report.html' for llm_report_html option
- llm_requests_per_minute value is set to 100

COVERAGE

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

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `llm_max_retries` option is not correctly set to its default value of 5, leading to incorrect configuration.

Key Assertions:

- The `llm_max_retries` option should be set to 9 when loading configuration from CLI.
- The `llm_max_retries` option should have a default value of 5.
- The test should fail if the `llm_max_retries` option is not set to 9 or has a different value than its 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	25 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282-283, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

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

AI ASSESSMENT

Scenario: Test loads values from ini options with mock Pytest configuration.

Why Needed: This test prevents a potential regression where the `load_config` function relies on `getini` to retrieve ini values, which may not be available or properly configured.

Key Assertions:

- cfg.provider == 'ollama'
- cfg.model == 'llama3'
- cfg.llm_context_mode == 'balanced'
- cfg.llm_requests_per_minute == 10
- cfg.llm_max_retries == 5
- cfg.report_html == 'report.html'
- cfg.report_json == '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 Config with aggregation settings to ensure correct directory and policy.

Why Needed: This test prevents a potential bug where the configuration is not set correctly for aggregation.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Tests Config with all output paths.

Why Needed: Prevents a potential bug where the test fails if any of the report files are missing or corrupted.

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `capture_failed_output` parameter of `Config` is set to `True`.

Why Needed: This test prevents a potential bug where the capture output is not properly handled when `capture_failed_output` is `False`.

Key Assertions:

- config.capture_failed_output is True
- assert config.capture_failed_output == True
- Verify that setting `capture_failed_output` to `False` does not prevent test execution.
- Test that the capture output is properly handled when `capture_failed_output` is `True`.
- Ensure that the assertion passes even if `capture_output_max_chars` is set to a lower value than 8000.
- Verify that the test can still pass without capturing any output.
- Check that the test does not raise an exception when `capture_failed_output` 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	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_options_extended.py::TestConfigAnnotations::test_compliance_settings 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the configuration of compliance settings.

Why Needed: This test prevents a potential bug where the configuration is not set correctly for compliance settings.

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: Prevents a bug where coverage settings are not properly configured, leading to incorrect test results.

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 that the `llm_context_exclude_globs` option is correctly excluded custom Python files and log files.

Why Needed: This test prevents a regression where the `llm_context_exclude_globs` option might be incorrectly or unexpectedly included in the list of exclude globs for custom Python files and log files.

Key Assertions:

- The string `*.pyc` should be present in the `llm_context_exclude_globs` list.
- The string `*.log` should be present in the `llm_context_exclude_globs` list.
- Custom Python files (e.g., `custom_file.py`) and log files (e.g., `custom_log.log`) should not be included in the `llm_context_exclude_globs` list.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `llm_context_include_globs` attribute includes the correct globs.

Why Needed: This test prevents a potential bug where the include globs are not correctly configured, potentially leading to incorrect or missing files being included in the LLM context.

Key Assertions:

- The `*.py` glob matches files with the `.py` extension.
- The `*.pyi` glob matches files with the `.pyi` extension.
- The `*.py` glob includes all Python source files (`.py`, `.pyi`) in the specified directory.
- The `*.pyi` glob includes all Python source files with the `.pyi` extension (e.g., `.pyc`) in the specified directory.
- Files not matching either `*.py` or `*.pyi` are excluded from the LLM context.
- The include globs are correctly configured to match the expected file types and directories.
- The test passes without any errors or warnings, indicating that the configuration is correct.

COVERAGE

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

PASSED tests/test_options_extended.py::TestConfigAnnotations::test_invocation_settings 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that `include_pytest_invocation` is set to `False` for the specified configuration.

Why Needed: Prevents a potential bug where the `include_pytest_invocation` setting is incorrectly configured, potentially leading to unexpected behavior or errors during testing.

Key Assertions:

- The `include_pytest_invocation` attribute of the test configuration object is set to `False`.
- The `include_pytest_invocation` attribute does not match the expected value of `False` for this specific configuration.
- The `include_pytest_invocation` attribute is correctly initialized with a boolean value.
- A pytest invocation setting is present in the test configuration, but it is not set to `True` or `False` as intended.
- The `invocation_redact_patterns` list contains a regular expression that matches API keys, which may be causing issues during testing.
- The `include_pytest_invocation` attribute is not being used in this specific test case.
- A different configuration object has the same `include_pytest_invocation` value as the one tested here.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Tests the configuration of LLM execution settings.

Why Needed: Prevents regression in LLMS when using a large number of tests or high concurrency.

Key Assertions:

- The value of llm_max_tests is set to 50.
- The value of llm_max_concurrency is set to 8.
- The value of llm_requests_per_minute is set to 12.
- The value of llm_timeout_seconds is set to 60 seconds.
- The value of llm_cache_ttl_seconds is set to 3600 seconds (1 hour).
- The cache directory is set to .cache.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `llm_include_param_values` parameter is set to `True` and that its maximum value is 200.

Why Needed: This test prevents a potential bug where the LLM param settings are not properly configured, potentially leading to incorrect output or errors.

Key Assertions:

- config.llm_include_param_values should be `True`
- config.llm_param_value_max_chars should be `200`

COVERAGE

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

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

AI ASSESSMENT

Scenario: Tests the configuration of LLM settings provided by OLLAMA.

Why Needed: This test prevents a potential bug where the model and context bytes are not correctly configured for optimal performance.

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 set to 20.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the `repo_root` attribute is set correctly to `/project`.

Why Needed: This test prevents a potential bug where the `repo_root` attribute is not set correctly, potentially leading to issues with repository configuration.

Key Assertions:

- config.repo_root
- is equal to Path('/project')
- config.repo_root is set to /project

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 `Config` class with valid include phase values.

Why Needed: Prevents a potential bug where invalid or missing include phases are passed to the `validate()` method, potentially causing validation errors.

Key Assertions:

- The `include_phase` attribute of each configuration object is not present in any error messages.
- No error messages are raised when an include phase value is valid (e.g., 'run', 'setup', or 'all').
- All included phases are correctly validated and do not cause any validation errors.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verify that the default exclude globs are correctly set for the LLM context.

Why Needed: This test prevents a potential bug where the default exclude globs do not include certain files or directories, potentially leading to unexpected behavior or errors.

Key Assertions:

- The function `Config().llm_context_exclude_globs` returns a list of strings that includes `*.pyc`, `__pycache__/*`, and `*secret*`, `*password*`.
- The test asserts the presence of these globs in the default exclude list.
- If any of the assert statements fail, it would indicate an issue with the default exclude globs being set correctly.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test default redact patterns for ConfigDefaultsMaximal test.

Why Needed: Prevents a potential bug where the default redact patterns are not correctly detected, potentially leading to incorrect configuration of the application.

Key Assertions:

- The '--password' and '--token' pattern names should be found in the invocation_redact_patterns list.
- The 'api[_-]?key' pattern name should be found in the invocation_redact_patterns list.
- Any other patterns that start with '--api_' or end with '_key' should also be present in the invocation_redact_patterns list.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test default values of the test_config_defaults_maximal module.

Why Needed: This test prevents regression in case the default configuration settings are not correctly set.

Key Assertions:

- The provider should be 'none'.
- The llm_context_mode should be 'minimal'.
- The llm_context_bytes should be 32000 bytes.
- The omit_tests_from_coverage flag should be True.
- The include_phase should be 'run'.

COVERAGE

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

PASSED `tests/test_options_maximal.py::TestConfigHelpersMaximal::test_is_llm_enabled` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify the correct enabled status of LLM for different providers.

Why Needed: This test prevents regression in LLM configuration settings.

Key Assertions:

- The `Config` object with provider 'none' should return False when `is_llm_enabled()` is called.
- The `Config` object with provider 'ollama' should return True when `is_llm_enabled()` is called.
- The `Config` object with provider 'litellm' should return True when `is_llm_enabled()` is called.
- The `Config` object with provider 'gemini' should return True when `is_llm_enabled()` is 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/options.py`	3 lines (ranges: 107, 147, 224)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_aggregate_policy` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the `validate` method of `Config` class for an invalid aggregate policy.

Why Needed: Prevents a potential bug where an invalid aggregate policy is not properly validated and returns an error message.

Key Assertions:

- The `validate` method should return exactly one error for an invalid aggregate policy.
- The error message should contain the string 'Invalid aggregate_policy 'invalid''.
- The error message should be present in the first error found during validation.

COVERAGE

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

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_context_mode` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the `validate` method of the `Config` class when an invalid context mode is provided.

Why Needed: This test prevents a potential bug where the `validate` method returns multiple errors for an invalid context mode, making it harder to identify and fix the issue.

Key Assertions:

- The `validate` method should return exactly one error message for an invalid context mode.
- The error message should contain 'Invalid llm_context_mode 'invalid''.
- The test should fail when an invalid context mode is provided.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	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 the `include_phase` parameter with an invalid value.

Why Needed: Prevents a potential bug where the test fails due to an incorrect or missing error message for an invalid `include_phase` value.

Key Assertions:

- The function `validate()` returns exactly one error message.
- The error message contains the string 'Invalid include_phase 'invalid'.
- The error message is not empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-205, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_provider` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test validates an invalid provider.

Why Needed: Prevents a potential bug where the test does not catch and report invalid providers.

Key Assertions:

- The function `validate()` should return exactly one error message for an invalid provider.
- The error message for an invalid provider should contain the exact phrase 'Invalid provider 'invalid''.
- The test should assert that there is only one error message in total.
- The first error message should be the entire string 'Invalid provider 'invalid'.
- The error message should not 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`	`19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_numeric_bounds` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_numeric_bounds

Why Needed: This test prevents regression of invalid numeric values in the config.

Key Assertions:

- The function `validate()` should return an error for invalid numeric values.
- The function `validate()` should throw an error if 'llm_context_bytes' is not a non-negative integer.
- The function `validate()` should throw an error if 'llm_max_tests' is negative.
- The function `validate()` should throw an error if 'llm_requests_per_minute' is zero.
- The function `validate()` should throw an error if 'llm_timeout_seconds' is zero.
- The function `validate()` should return at least 4 errors for invalid numeric values.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`21 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209-217, 220)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_valid_config` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that an invalid configuration returns an empty list.

Why Needed: Prevents a potential bug where the function does not handle all possible input configurations correctly.

Key Assertions:

- The `validate` method of the `Config` class should return an empty list for any valid configuration.
- Any invalid configuration should raise an exception or return a meaningful error message.
- The function should be able to handle different types of input configurations (e.g., dictionaries, lists).
- If the input is not a dictionary or list, the function should raise an `TypeError` or return an error message.
- The function should check for any invalid keys in the configuration and raise an exception if found.
- Any missing required keys should be raised with an appropriate error message.
- The function should handle nested configurations correctly (if applicable).
- If the input is not a valid dictionary or list, the function should return a meaningful error message.

COVERAGE

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

PASSED `tests/test_plugin_integration.py::TestPluginConfigLoading::test_config_defaults` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `Config` object has default settings.

Why Needed: This test prevents a potential bug where the configuration is missing or incorrectly configured due to missing plugin options.

Key Assertions:

- The `cfg` variable should be an instance of `Config`.
- The value of `cfg` should not be None.
- The type of `cfg` should be `Config`.

COVERAGE

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

Why Needed: Prevent a potential bug where the plugin configuration is inaccessible due to incorrect import or setup.

Key Assertions:

- The `pytestconfig` object should be an instance of `pytest.config.Config`.
- The `pytestconfig` object should not be `None`.
- The `pytestconfig` object's attributes (e.g. `markers`, `options`) should be accessible.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- assert True is executed without any errors.
- assert False is executed with an error message indicating context marker issue.
- assert 'Context marker should not cause errors.' is printed to the console.
- assert 'LLM marker' or 'context override' is present in the test output.
- assert 'Opt-out, context override' is present in the test output.
- assert 'Captured stdout/stderr for failed tests (opt-in)' is present in the test 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)

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: The 'requirement_marker' function is being tested to ensure it does not throw any errors.

Why Needed: This test prevents a potential bug where the 'requirement_marker' function could be used in error conditions, causing unexpected behavior or crashes.

Key Assertions:

- The 'requirement_marker' function should not raise an exception when called with no arguments.
- The 'requirement_marker' function should not throw any errors when executed without any inputs.
- The 'requirement_marker' function should not cause any runtime errors when used correctly and in the correct 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` 32ms 🛡️ 6

AI ASSESSMENT

Scenario: Test the integration of report writer with pytest_llm_report.

Why Needed: This test prevents regression when integrating report writer into pytest_llm_report, ensuring that full report generation flow works correctly.

Key Assertions:

- Verify existence of 'report.json' file in tmp_path.
- Verify correct number of passed tests (1) and failed tests (1) in the JSON data.
- Verify presence of 'test_a.py' and 'test_b.py' in the HTML 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	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.

Why Needed: The test prevents a regression where pytest_collectreport fails to report the plugin's collection even though it was disabled.

Key Assertions:

- mocked session.config.stash.get._enabled_key returns False
- pytest_collectreport() does not call _enabled_key with True
- mock_report.session.config.stash.get.asserts_called_with(_enabled_key, False) is False

COVERAGE

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

Why Needed: This test prevents a potential regression where `pytest_collectreport` does not call the collector when collectreport is enabled.

Key Assertions:

- The `pytest_collectreport` function should be able to find and stash the `_collector_key` key in the session configuration.
- The `stash_get` function should return `True` for the `_enabled_key` key.
- The `handle_collection_report` method of the collector should be called with the stash value containing the `_collector_key` key.
- The `handle_collection_report` method of the collector should not throw an exception if it cannot find the stash value.
- The `stash_get` function should return `True` for all other keys in the session configuration.
- The `pytest_collectreport` function should be able to handle a mock report object correctly.
- The `mock_collector.handle_collection_report` call should not throw an exception if it cannot find the stash 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`	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 `pytest_collectreport` does not raise an exception when no session is available.

Why Needed: Prevent regression in plugin behavior when a test has no active session.

Key Assertions:

- The function `pytest_collectreport(mock_report)` should not be called with a `session` attribute that is missing.
- No exception should be raised if the `session` attribute is deleted from the mock report object.
- The `session` attribute of the mock report object should be `None` after deletion.
- The function `pytest_collectreport(mock_report)` should not raise an AttributeError when the session attribute is accessed.
- The function `pytest_collectreport(mock_report)` should not raise a TypeError if the session attribute is accessed with no arguments.
- The function `pytest_collectreport(mock_report)` should not raise a KeyError if the session attribute is accessed without checking for its existence 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/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: Tests the behavior of `pytest_collectreport` when a session is `None`.

Why Needed: Prevents potential `pytest_collectreport` exceptions that may occur when trying to collect reports for an empty session.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	8 lines (ranges: 387-388, 391, 395-397, 408, 412)

PASSED tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_llm_enabled_warning 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that LLM enabled warning is raised when using the Ollama LLM report provider.

Why Needed: To prevent a potential bug where the LLM report provider 'ollama' is enabled and raises an error when configured with pytest.

Key Assertions:

- The `pytest_llm_report_provider` option should be set to `'ollama'`.
- The `llm_report_html`, `llm_report_json`, `llm_report_pdf`, `llm_evidence_bundle`, `llm_dependency_snapshot`, `llm_requests_per_minute`, `llm_aggregate_dir`, `llm_aggregate_policy`, `llm_aggregate_run_id`, and `llm_aggregate_group_id` options should be set to `None`.
- The `llm_max_retries` option should also be set to `None`.
- The `rootpath` and `stash` options should not be set.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	44 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	29 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 387-388, 391, 395-397)

PASSED tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_validation_errors 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that validation errors raise UsageError when setting invalid config.

Why Needed: Prevents a potential bug where the plugin does not handle configuration errors properly and raises a UsageError instead of providing meaningful error messages.

Key Assertions:

- Mocking `getini` method with an invalid key returns a dictionary containing 'llm_report_provider' as the only valid value.
- Setting `option.llm_report_html`, `option.llm_report_json`, `option.llm_report_pdf`, `option.llm_evidence_bundle`, `option.llm_dependency_snapshot`, `option.llm_requests_per_minute`, `option.llm_aggregate_dir`, `option.llm_aggregate_policy`, `option.llm_aggregate_run_id`, and `option.llm_aggregate_group_id` to None.
- Setting `llm_report_provider` to an invalid value raises a ValueError with the message 'configuration errors'.
- The `pytest_configure` function is called with a valid config object, but it does not raise a UsageError.
- Calling `pytest_configure` with an invalid config object should raise a UsageError.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	43 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 248, 251-253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	25 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-199, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_worker_skip` 1ms 🛡 2

AI ASSESSMENT

Scenario: Test that configure skips on xdist workers.

Why Needed: This test prevents a potential regression where the plugin might skip configuration due to an incorrect or incomplete worker input.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`17 lines (ranges: 169-171, 173-175, 177-179, 183-184, 387-388, 391, 395-397)`

PASSED `tests/test_plugin_maximal.py::TestPluginConfigureFallback::test_pytest_configure_fallback_load` 3ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that fallback to load_config occurs when Config.load is missing in pytest_configure function.

Why Needed: Prevents a potential bug where the test fails due to missing Config.load method call.

Key Assertions:

- The Config.load() method should be called with no arguments.
- The Config.validate() method should return an empty list.
- The load_config() function should not be called directly.
- The option.llm_report_html and option.llm_max_retries attributes should remain unchanged.
- The option.llm_max_retries attribute should still be None after the test.
- The mock_load object returned by patch should have been called once with no arguments.
- The Config.load() method call should not raise an exception.

COVERAGE

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

Why Needed: This test prevents a potential bug where the plugin fails to load configuration due to missing or invalid INI values.

Key Assertions:

- The `llm_report_provider` option is set correctly.
- The `llm_report_model` option is set correctly.
- The `llm_report_context_mode` option is set correctly.
- The `llm_report_requests_per_minute` option is set to 10.
- The `report_html` option is set to 'ini.html'.
- The `report_json` option is set to 'ini.json'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	31 lines (ranges: 107, 147, 248, 251-263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)

PASSED tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_cli_overrides_ini 2ms 🛡️ 3

AI ASSESSMENT

Scenario: Test CLI options override INI options.

Why Needed: This test prevents regression where the CLI options override INI options, ensuring that the correct report files are used even when the INI file is not present or does not contain the required configuration.

Key Assertions:

- cfg.report_html == 'cli.html'
- cfg.report_json == 'cli.json'
- cfg.report_pdf == 'cli.pdf'
- cfg.report_evidence_bundle == 'bundle.zip'
- cfg.report_dependency_snapshot == 'deps.json'
- cfg.llm_requests_per_minute == 20
- cfg.aggregate_dir == '/agg'
- cfg.aggregate_policy == 'merge'
- cfg.aggregate_run_id == 'run-123'
- cfg.aggregate_group_id == '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	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 terminal summary is not displayed when the plugin is disabled.

Key Assertions:

- Mocked stash.get() with _enabled_key and False returns None, as expected.
- Mocked stash.get() with _enabled_key and True does not return None, as expected.
- Mocked stash.get() with _enabled_key and an incorrect value (e.g., 1) should have been called once.
- The stash.get method is called only once, even if the plugin is enabled multiple times.
- The stash.get method is not called when the plugin is disabled or does not exist.
- The stash.get method returns None instead of raising an exception when it fails to get a value.
- The stash.get method should raise an exception when it fails to get a value, as expected.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	9 lines (ranges: 238, 242-243, 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::test_terminal_summary_worker_skip` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that terminal summary skips on xdist worker when configured to skip.

Why Needed: This test prevents a regression where the plugin does not properly handle skipping in xdist workers.

Key Assertions:

- The `pytest_terminal_summary` function should return None for the given configuration.
- The `workerinput` attribute of the mock config object is set to 'gw0'.
- No output is produced from the test (i.e., no output is printed or displayed).

COVERAGE

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

Why Needed: This test prevents a potential bug where the report HTML is not set correctly if the `pytest_llm_report` CLI option or INI file does not provide an `llm_report_html` value.

Key Assertions:

- cfg.report_html == 'out.html'
- mock_config.option.llm_report_html == 'out.html'
- cfg.getini('llm_report_html') == None
- cfg.rootpath == '/root'

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	36 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270-283, 286-295, 298, 300)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_disabled` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test makereport skips when disabled.

Why Needed: This test prevents a regression where the plugin does not report any issues even though it should.

Key Assertions:

- mock_item.config.stash.get returns False
- mock_call is called with no arguments
- mock_outcome.get_result returns a MagicMock object
- gen.send raises StopIteration and passes mock_outcome
- gen.next yields the mock call to the next hookwrapper

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	7 lines (ranges: 387-388, 391-392, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_enabled` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test makereport calls collector when enabled.

Why Needed: This test prevents a potential regression where the collector is not called when makereport is enabled.

Key Assertions:

- The `mock_collector` should be called with the `mock_report` argument when `makereport` is enabled.
- The `mock_collector` should have been called once with the `mock_report` argument.
- The `mock_collector` should not have been called without any arguments (i.e., when makereport is disabled).
- The `mock_collector` should be called before calling `next(gen)` to ensure it has a chance to handle the log report.
- The `mock_collector` should not be called after `next(gen)` because there's no need to handle the log report again.
- The `mock_collector` should have been called with the correct arguments (i.e., `mock_report`) when `makereport` 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/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_disabled` 1ms 🛡 2

AI ASSESSMENT

Scenario: Test that collection_finish is skipped when disabled for Pytest.

Why Needed: To prevent a regression where the plugin's hooks are not executed correctly when collection_finish is disabled.

Key Assertions:

- mock_session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_collection_finish(mock_session) should be called with _enabled_key as 'pytest_collection_finish' and False as its argument
- mock_session.config.stash.get.return_value should not be True
- _enabled_key should be 'pytest_collection_finish'
- pytest_collection_finish should not be called with any arguments

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 387-388, 391, 395-397, 431-432)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_enabled` 2ms 🛡 2

AI ASSESSMENT

Scenario: Verify that collection_finish is called when collection finish is enabled.

Why Needed: Prevents a potential bug where the collector does not get notified about the finished items in Pytest.

Key Assertions:

- mock_collector.handle_collection_finish.assert_called_once_with(mock_session.items)
- mock_session.items[0].is_pytest_collection_finished() == False
- mock_session.items[1].is_pytest_collection_finished() == 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	10 lines (ranges: 387-388, 391, 395-397, 431, 435-437)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_disabled` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that sessionstart skips when disabled and verifies the expected behavior.

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

Key Assertions:

- The `pytest_sessionstart` function should be called with the `_enabled_key` parameter set to `False`.
- The `get` method of the stash configuration should have been called with the `_enabled_key` parameter and a boolean value of `False`.
- The `assert_called_with` method of the mock session's `config.stash.get` method should have been called.
- The `pytest_sessionstart` function should not be called with any other parameters.
- The `get` method of the stash configuration should return a boolean value of `False`.
- The `assert_called_with` method of the mock session's `config.stash.get` method should have been called with `_enabled_key` and `False` as arguments.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 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 when pytest_sessionstart is called with an enabled configuration.

Key Assertions:

- The key _collector_key should be present in the mock stash.
- The key _start_time_key should be present in the mock stash.
- A MagicMock object with type _collector_key and _start_time_key should be created.
- A MagicMock object with type _enabled_key should be set to True in the stash_dict.
- _enabled_key should have a value of True in the stash_dict.
- The pytest_sessionstart function should create a collector when called with an enabled configuration.
- The collector should be initialized correctly, i.e., it should contain both _collector_key and _start_time_key.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	11 lines (ranges: 387-388, 391, 395-397, 448, 452, 455, 457-458)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test pytest_addoption adds expected arguments to the parser.

Why Needed: pytest_addoption prevents a potential bug where it does not add all required options to the parser.

Key Assertions:

- Verify that `--llm-report` is added as an option.
- Verify that `--llm-coverage-source` is also added as an option.
- Check if both options are present in the parsed arguments.
- Verify that the group name matches 'llm-report' and the report type matches 'LLM-enhanced test reports'.
- Ensure that the `addoption` method of the group returns a tuple with two elements, where the first element is the option string.

COVERAGE

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

Key Assertions:

- The function `pytest_addoption(parser)` is called with a `MagicMock` object as the parser.
- The `addini.call_args_list` attribute of the parsed parser contains lines starting with 'llm_report_'.
- The string 'llm_report_html' is found in the ini calls.
- The string 'llm_report_json' is found in the ini calls.
- The string 'llm_report_max_retries' is found 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: This test prevents regression in coverage percentage calculation logic when using the terminal summary feature.

Key Assertions:

- The `report_html` option is set to 'out.html' and the `CoverageMapper` class is mocked with a mock `Coverage` object.
- The `MockStash` dictionary is created with the provided stash data.
- The `patch` decorator ensures that the `pathlib.Path.exists` function returns True when it should return False, simulating coverage file existence.
- The `patch` decorator ensures that the `Coverage` class is mocked with a mock object and its methods are called correctly.
- The `MockStash` dictionary is loaded into the `CoverageMapper` instance using the `load` method.
- The `report` method of the `Coverage` object is called to generate the coverage report, which should return 85.5 as expected.
- The test verifies that the coverage percentage calculation logic works correctly by checking if the reported coverage matches the expected value.

COVERAGE

`src/pytest_llm_report/collector.py`	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
`src/pytest_llm_report/options.py`	3 lines (ranges: 107, 147, 224)
`src/pytest_llm_report/plugin.py`	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_llm_enabled` test passes without any errors.
- Check if the correct configuration is passed to `pytest_terminal_summary`.
- Ensure that the LLM annotator is called with the provided config.
- Verify that the annotation is performed correctly and returns the expected result.
- Confirm that the report writer is patched correctly and can write the output.
- Test that the coverage mapper is not used in this scenario.
- Verify that the `pytest_llm_report.llm.annotator.annotate_tests` function is called only once.
- Check if the correct model name is retrieved from the provider.

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 even when it is supposed to be missing.

Key Assertions:

- ...
- ...
- ...
- Mocking stash with only enabled key and no config key should not trigger the creation of a collector.
- Mocking stash with only enabled key but no config key should still create a collector.
- Mocking stash with only disabled key and no config key should not trigger the creation of a collector.
- ...
- ...
- ...

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: Prevents a regression where the plugin does not aggregate terminal summaries correctly when report_html and report_json are set to out.html and out.json respectively.

Key Assertions:

- The stash object returned by pytest_terminal_summary should support both get() and [] indexing.
- The aggregation function should be called once with the correct arguments (0, stash).
- The ReportWriter should write JSON and HTML files correctly.
- The aggregation report should contain the expected 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/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 number of files.

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 it encounters a file that cannot be loaded.
- The `coverage.Coverage` object returned by `CoverageMapper` should contain the correct coverage statistics for all files.
- The `report_writer.ReportWriter` instance should not throw an error when writing to the report HTML file.
- The `pytest_terminal_summary` function should return a summary with the expected coverage statistics even if it encounters an error during load.
- The `pytest_llm_report.report_writer.ReportWriter` instance should be able to write the report without raising an exception.
- The `pytest_llm_report.options.Config` object returned by `Config` should contain the correct configuration settings for coverage calculation.
- The `pytest_terminal_summary` function should return a summary with the expected error message when it encounters an OSError during load.

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 the llm_context_mode is set to 'balanced' and the test file contains unbalanced dependencies.

Key Assertions:

- The 'utils.py' file should be present in the assembled context.
- The 'def util()' function should be found in the 'utils.py' file within the assembled context.
- The line ranges of '1-2' and line count of '2' should match the coverage entry for 'utils.py'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/prompts.py`	`51 lines (ranges: 33, 49, 52, 55, 58, 60-61, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 116, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-155, 158-159, 163, 191-192, 194)`

PASSED `tests/test_prompts.py::TestContextAssembler::test_assemble_complete_context` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Assembling a complete context for the 'test_a.py' test file and verifying that the 'test_1' function is included in the source code.

Why Needed: This test prevents regression where the 'test_a.py' test file is not properly assembled into a complete context, potentially leading to incorrect test results or errors.

Key Assertions:

- The 'test_1' function should be present in the source code of the assembled context.
- The 'test_1' function should be included in the assembly output.
- The 'test_a.py::test_1' path should match the expected location in the assembled context.
- The 'test_a.py::test_1' function name should be present in the source code of the assembled context.
- The 'test_a.py::test_1' function signature should match the expected signature in the assembled context.
- The assembly output should contain the 'test_1' function definition.
- The assembly output should not contain any errors or warnings related to the 'test_1' function.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	34 lines (ranges: 33, 49, 52, 55, 58, 60, 63, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 116, 132-133, 180)

PASSED `tests/test_prompts.py::TestContextAssembler::test_assemble_minimal_context` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Verifies that the ContextAssembler can assemble a minimal context for a test file with a single test function.

Why Needed: This test prevents regression when using the 'minimal' llm_context_mode, as it ensures the assembler only creates a minimal context without any additional dependencies.

Key Assertions:

- The source code of the test file is present in the assembled context.
- The context contains no references to external modules or functions.
- The test function `test_1` is present in the assembled context.
- The context has an empty dictionary as its value.
- No additional dependencies are included in the context.
- No external imports are used in 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/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: Tests the ContextAssembler with balanced context limits to ensure it correctly truncates long files.

Why Needed: This test prevents a potential bug where the ContextAssembler does not truncate long files according to the specified context limit.

Key Assertions:

- ...
- ...
- ContextAssemblyResult: The assembled context contains only the truncated part of the file.
- The length of the assembled context is within the allowed limit (40 bytes).
- The file name in the assembled context does not exceed the specified limit (20 bytes + truncation message).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	34 lines (ranges: 33, 49, 52, 55, 58, 60-61, 65, 78-79, 82-84, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-156, 158-159, 163, 191-192, 194)

PASSED `tests/test_prompts.py::TestContextAssembler::test_get_test_source_edge_cases` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the ContextAssembler with edge cases where a non-existent file is provided.

Why Needed: This test prevents potential issues when using the ContextAssembler with files that do not exist, as it ensures the assembler correctly handles such scenarios.

Key Assertions:

- The function `_get_test_source` returns an empty string for a non-existent file.
- The function `_get_test_source` includes the full path of the test file in its output.
- The function `_get_test_source` correctly identifies the 'test' keyword in the nested test name with parameters.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/prompts.py	26 lines (ranges: 33, 78-79, 82-84, 86-87, 92, 94-95, 98-101, 103-112, 116)

PASSED `tests/test_prompts.py::TestContextAssembler::test_should_exclude` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the ContextAssembler should exclude certain files from being processed by the LLM.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly excludes important files, leading to incorrect results or errors.

Key Assertions:

- The file 'test.pyc' is excluded because it has been compiled and its contents are not relevant to the LLM's processing.
- The file 'secret/key.txt' is excluded because it contains sensitive information that should be protected by the ContextAssembler.
- The file 'public/readme.md' is included in the exclusion list because it does not contain any sensitive or confidential information.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`1 lines (ranges: 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/prompts.py`	`5 lines (ranges: 33, 191-194)`

PASSED `tests/test_ranges.py::TestCompressRanges::test_consecutive_lines` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The 'compress_ranges' function is tested to ensure consecutive lines are compressed correctly.

Why Needed: This test prevents a potential bug where consecutive lines in the input list do not use range notation.

Key Assertions:

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

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)`
`src/pytest_llm_report/util/ranges.py`	`12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)`

PASSED `tests/test_ranges.py::TestCompressRanges::test_duplicates` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the function correctly handles duplicate ranges.

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

Key Assertions:

- assert compress_ranges([1, 2, 2, 3, 3, 3]) == '1-3'
- assert compress_ranges([4, 5, 6, 7, 8, 9]) == None
- assert compress_ranges([1, 1, 2, 2, 3, 3]) == '1-3'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

PASSED `tests/test_ranges.py::TestCompressRanges::test_empty_list` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Testing the `compress_ranges` function with an empty input list.

Why Needed: This test prevents a potential bug where an empty list is not correctly compressed to an empty string.

Key Assertions:

- The function should return an empty string for an empty input list.
- The function should handle the case of an empty list without raising any exceptions or errors.
- The output of `compress_ranges([])` should be a string representing an empty range (e.g., '[]') rather than None or some other value.
- Any additional ranges in the input list should not affect the output of `compress_ranges([])`.
- The function should handle lists containing multiple ranges correctly (e.g., `[1, 2]` and `[3, 4]`).
- If a non-empty range is present in the input list, it should be included in the compressed result.
- The function should not return an empty string for a list with one or more ranges (e.g., `[1, 5]` and `[2, 6]`).

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/ranges.py`	2 lines (ranges: 29-30)

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

AI ASSESSMENT

Scenario: Test the function when given a mixed range of values.

Why Needed: This test prevents regression in cases where ranges are present alongside single values.

Key Assertions:

- The function should correctly group the ranges and singles into separate output strings.
- The function should handle ranges with inclusive or exclusive endpoints correctly.
- The function should not include any invalid range combinations (e.g., -1-3, etc.).
- The function should preserve the original order of elements within each range.
- The function should handle edge cases where there are no values in a range.
- The function should correctly identify ranges with overlapping endpoints.
- The function should not include any invalid or duplicate output strings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	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: The 'test_non_consecutive_lines' test verifies that non-consecutive lines are comma-separated.

Why Needed: This test prevents a regression where consecutive lines are not separated by commas.

Key Assertions:

- The input list should be sorted in ascending order.
- Each element in the list should be an integer.
- Non-consecutive elements should be separated by commas.
- Consecutive elements should be separated by spaces.
- All elements should be comma-separated, with no leading or trailing spaces.
- No non-integer values should be present in the input list.
- The output string should match the expected result.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	14 lines (ranges: 29, 33, 35-37, 39-40, 45-47, 50, 52, 65-66)

PASSED `tests/test_ranges.py::TestCompressRanges::test_single_line` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The 'single_line' test verifies that the `compress_ranges` function does not attempt to compress a single-line string.

Why Needed: This test prevents regression when the `compress_ranges` function is called with a single-element list of integers, as it may incorrectly try to apply range notation.

Key Assertions:

- The input list should be empty or contain only one element.
- The output should be the original string '5'.
- No error message or exception should be raised.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/ranges.py`	10 lines (ranges: 29, 33, 35-37, 39, 50, 52, 65-66)

PASSED `tests/test_ranges.py::TestCompressRanges::test_two_consecutive` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: tests/test_ranges.py::TestCompressRanges::test_two_consecutive

Why Needed: This test prevents a potential bug where two consecutive numbers are not compressed to range notation.

Key Assertions:

- The function compresses the input list into '1-2' if it contains two consecutive numbers.
- If the input list does not contain two consecutive numbers, the function should return an error message or raise a meaningful exception.
- The function should handle edge cases where the input list is empty or only contains one number correctly.
- Two consecutive numbers in the input list should be treated as separate ranges.
- Non-consecutive numbers in the input list should not be compressed to range notation.
- The function should raise a meaningful exception when given an invalid input, such as non-numeric values or out-of-range numbers.
- If the input is a single number, the function should return that number instead of trying to compress it into a range.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

PASSED `tests/test_ranges.py::TestCompressRanges::test_unsorted_input` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'test_unsorted_input' verifies that the function handles unsorted input correctly.

Why Needed: The test prevents a potential bug where the function would incorrectly group ranges in an unsorted list.

Key Assertions:

- Input is sorted before compression.
- Ranges are grouped by their correct order (e.g., '1-3' for [1, 2, 3]).
- Ranges with the same start value but different end values are correctly separated ('5').

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/ranges.py`	16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)

PASSED `tests/test_ranges.py::TestExpandRanges::test_empty_string` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Testing the `expand_ranges` function with an empty string.

Why Needed: Prevents a potential bug where the function returns incorrect results for empty input strings.

Key Assertions:

- Input string is empty (length: 0).
- Function should return an empty list (`[]`).

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/ranges.py`	2 lines (ranges: 81-82)

PASSED `tests/test_ranges.py::TestExpandRanges::test_mixed` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: test_mixed verifies the expansion of mixed range values.

Why Needed: This test prevents a regression where the function incorrectly expands single numbers into ranges.

Key Assertions:

- The input string should be expanded correctly to include all specified numbers in their respective ranges.
- Single numbers should not be included in the output if they do not fall within any range.
- Ranges should include all specified values without gaps or overlaps.
- Invalid characters (e.g., commas) should not affect the expansion of the input string.
- The function should handle edge cases where a single number is at the start/end of a range correctly.
- The function should handle ranges with overlapping values correctly.
- The function should raise an error for invalid input strings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	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 called with a string argument containing a range (e.g., '1-3') and it returns the expected result.

Why Needed: This test prevents a potential bug where the function does not expand ranges correctly, potentially leading to incorrect results or errors.

Key Assertions:

- The input string should be in the format 'start-end' (e.g., '1-3')
- The function should return a list of numbers from start to end (inclusive)
- The start value should be less than or equal to the end value
- All values in the range should be integers
- No negative numbers should be allowed in the range
- The function should handle edge cases where the input string is empty or contains only one 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/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 `compress_ranges` and `expand_ranges` functions should be able to reverse their operations.

Why Needed: This test prevents a potential regression where the order of elements in the compressed or expanded ranges is not preserved.

Key Assertions:

- original = [1, 2, 3, 5, 10, 11, 12, 15]
- compressed = compress_ranges(original)
- expanded = expand_ranges(compressed)
- assert expanded == original

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/ranges.py	27 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67, 81, 84-91, 93, 95)

PASSED `tests/test_ranges.py::TestExpandRanges::test_single_number` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `expand_ranges` function returns a list containing only one element when given a single number.

Why Needed: This test prevents regression where the function incorrectly expands multiple numbers into lists with multiple elements.

Key Assertions:

- The input string should be a single number (e.g., '5')
- The output list should contain only one element (i.e., [5])

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/ranges.py`	7 lines (ranges: 81, 84-87, 93, 95)

PASSED `tests/test_render.py::TestFormatDuration::test_milliseconds` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the function formats duration as milliseconds for less than one second.

Why Needed: Prevents a potential bug where the function does not correctly format durations for values less than 1s.

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 function formats duration values correctly for seconds.

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

Key Assertions:

- {'message': 'Format should be in seconds', 'description': 'The format of the output is correct'}
- {'message': "Output should contain 's' suffix", 'description': "The output contains a 's' suffix for durations greater than or equal to 1 second"}
- {'message': 'No negative duration values are supported', 'description': 'Negative duration values are not supported by this 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/render.py`	2 lines (ranges: 65-66)

PASSED `tests/test_render.py::TestOutcomeToCssClass::test_all_outcomes` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: All outcomes should map to CSS classes.

Why Needed: This test prevents a potential CSS class mismatch issue where 'all' outcomes are incorrectly mapped to non-existent or incorrect classes.

Key Assertions:

- outcome-to-css_class('passed') == 'outcome-passed'
- outcome-to-css_class('failed') == 'outcome-failed'
- outcome-to-css_class('skipped') == 'outcome-skipped'
- outcome-to-css_class('xfailed') == 'outcome-xfailed'
- outcome-to-css_class('xpassed') == 'outcome-xpassed'
- outcome-to-css_class('error') == '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 case where 'outcome' is unknown in `outcome_to_css_class` function.

Why Needed: This test prevents a potential bug where the function returns an unexpected class for an unknown outcome.

Key Assertions:

- The function should return the default class 'outcome-unknown' when given an unknown outcome.
- The function should not raise any exceptions or errors when given an unknown outcome.
- The function should correctly handle cases where the outcome is not recognized by the `outcome_to_css_class` mapping.
- The function's output should be consistent with the expected default class for unknown outcomes.
- The function's behavior should not change between different Python versions or environments.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	8 lines (ranges: 79-85, 87)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_basic_report` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the report renders a complete HTML document with the expected elements.

Why Needed: This test prevents regression where the report does not render correctly due to missing or incorrect plugin and repository information.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	52 lines (ranges: 65-67, 79-85, 87, 121-124, 126-127, 131-132, 141-143, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_coverage` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test renders coverage to ensure that the rendered HTML includes the specified file and line count.

Why Needed: This test prevents regression by ensuring that the rendered HTML includes the expected information, which is critical for maintaining code coverage.

Key Assertions:

- The rendered HTML should include the specified file (src/foo.py).
- The rendered HTML should include the correct number of lines (5).

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/render.py`	`52 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-129, 131-132, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)`

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_llm_annotation` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the LLM annotation is included in the rendered HTML report.

Why Needed: This test prevents a potential security vulnerability where an attacker could manipulate the LLM annotations to bypass authentication.

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 LlmAnnotation object should contain a scenario attribute with value 'Tests login flow'.
- The LlmAnnotation object should contain a why_needed attribute with value 'Prevents auth bypass'.
- The html string should contain both 'Tests login flow' and 'Prevents auth bypass' strings.

COVERAGE

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

Why Needed: This test prevents a regression where the source coverage information is not included in the rendered HTML report, potentially misleading users about the code's coverage status.

Key Assertions:

- The 'Source Coverage' section should be present in the rendered HTML report.
- The 'src/foo.py' file path should be included in the 'Source Coverage' section.
- The source coverage percentage should be displayed as '80.0%' in the rendered HTML report.
- All statements in the source code should be covered by at least 8% of the total statements in the test.
- At least 2 statements in the source code should be missed, which corresponds to 20% of the total statements in the test.
- The 'covered_ranges' and 'missed_ranges' sections should contain ranges that accurately represent the covered and missed statements respectively.
- All statements in the source code should be covered by at least 80% of the total statements in the test.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/render.py`	63 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-127, 131-132, 141-142, 145-153, 158-164, 166-172, 177, 192, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_xpass_summary` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the rendered summary includes both 'XFailed' and 'XPassed' entries.

Why Needed: This test prevents a regression where the summary is missing 'XFailed' or 'XPassed' entries when using XPass tests.

Key Assertions:

- The HTML string contains the exact text 'XFailed', indicating that it was included in the rendered summary.
- The HTML string contains the exact text 'XPassed', indicating that it was included in the rendered summary.
- The presence of both 'XFailed' and 'XPassed' entries is confirmed by the assertion 'XFailed' being present in the HTML string.
- The absence of either 'XFailed' or 'XPassed' entry is not detected by this test.
- The test verifies that the rendered summary includes only one of each status, as expected for XPass 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/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 computes SHA-256 for different input strings and verifies the output is different.

Why Needed: This test prevents a bug where the same input string produces the same hash value, potentially leading to incorrect reporting or analysis of data.

Key Assertions:

- The function `compute_sha256` correctly computes the SHA-256 hash for both input strings.
- The output of `compute_sha256(b'hello')` is different from the output of `compute_sha256(b'world')`.
- The computed hash values are unique and cannot be compared directly.
- The function handles non-string inputs correctly (e.g., bytes, integers).
- The test case covers a common edge case where input strings are different but have the same content.
- The output of `compute_sha256(b'hello')` is not equal to `compute_sha256(b'world')` even though they contain different characters.
- The function correctly handles duplicate input strings, as expected for different content.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

PASSED `tests/test_report_writer.py::TestComputeSha256::test_empty_bytes` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'test_empty_bytes' verifies that an empty bytes input produces consistent hash.

Why Needed: Prevents a potential bug where an empty bytes input could lead to inconsistent hashes due to the SHA-256 algorithm's behavior with zero-length inputs.

Key Assertions:

- Input should be an empty bytes object (b'')
- Hash of the input should be equal to hash of another identical input (hash1 == hash2)
- Length of the hash output should match the expected SHA-256 hex length (64)

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 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` 4ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the `build_run_meta` method of ReportWriter with a test case that 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 test should be 60 seconds.
- The `pytest_version` attribute of the meta object should have a value.
- The `plugin_version` attribute of the meta object should be set to '0.1.0'.
- The `python_version` attribute of the meta object should also be set to a valid Python 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/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 the ReportWriter's ability to build a summary of all outcome types.

Why Needed: This test prevents regression where the `total` count is not accurate due to missing or incorrect outcome counts.

Key Assertions:

- The total number of outcomes should be equal to the sum of passed, failed, skipped, xfailed, xpassed, and error outcomes.
- Each outcome type (passed, failed, skipped, xfailed, xpassed, error) should appear exactly once in the summary.
- The `total` count should match the expected value based on the number of tests with each outcome type.
- The `passed`, `failed`, `skipped`, `xfailed`, and `xpassed` counts should be accurate for each test.
- The `error` count should also be accurate for each test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	19 lines (ranges: 156-158, 312, 314-315, 317-328, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_build_summary_counts` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the `total` count of outcomes is accurate.

Why Needed: This test prevents a regression where the total count of outcomes might be incorrect due to missing or incomplete tests.

Key Assertions:

- summary.total should equal 4 (the number of tests passed)
- summary.passed should equal 2 (the number of tests that passed)
- summary.failed should equal 1 (the number of tests that failed)
- summary.skipped should equal 1 (the number of tests that were skipped)

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/report_writer.py`	13 lines (ranges: 156-158, 312, 314-315, 317-322, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_create_writer` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the functionality of creating a ReportWriter instance.

Why Needed: To prevent a potential bug where the Writer's configuration or artifacts are not properly initialized.

Key Assertions:

- The `config` attribute of the `ReportWriter` instance should be equal to the provided `Config` object.
- The `warnings` list of the `ReportWriter` instance should be empty.
- The `artifacts` list of the `ReportWriter` instance should be empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	3 lines (ranges: 156-158)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_assembles_tests` 4ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that ReportWriter writes a report with all tests.

Why Needed: This test prevents regression where the report does not include all tests, potentially leading to incomplete or misleading reports.

Key Assertions:

- The length of the report.tests list should be equal to 2.
- The total 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` 5ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class writes a report with a total coverage percentage.

Why Needed: This test prevents regression where the coverage percentage is not included in the report.

Key Assertions:

- The `coverage_total_percent` attribute of the `report.summary` object should be equal to the provided `coverage_percent` value.
- The `report.summary.coverage_total_percent` property should have a numerical value.
- The coverage percentage should be calculated correctly based on the report's data.
- The test should fail if the coverage percentage is not included in the report.
- The coverage percentage should be reported as a percentage value (e.g., 85.5%).
- The `report.summary.coverage_total_percent` property should update correctly after writing the report.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	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 4ms 🛡️ 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/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` 4ms 🛡️ 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 in case where multiple tests have overlapping code coverage.

Key Assertions:

- The first test's coverage should be a single entry with file path 'src/foo.py'.
- The first test's coverage should only contain the specified file path.
- The number of entries in the report's first test's coverage should match the number of tests.
- The first test's coverage should have exactly one file path.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	94 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186-189, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330)

PASSED tests/test_report_writer.py::TestReportWriterWithFiles::test_atomic_write_fallback 5ms 🛡 5

AI ASSESSMENT

Scenario: Test that the ReportWriterWithFiles class falls back to direct write if an atomic write fails and the 'os.replace' mock is used.

Why Needed: This test prevents a regression where the atomic write operation fails, but instead of raising an exception, it falls back to direct write.

Key Assertions:

- The file 'report.json' should exist at the expected location.
- Any warnings with code 'W203' (indicating a permission issue) should be present in the report.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	67 lines (ranges: 229-231, 233, 235, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	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 5ms 🛡️ 5

AI ASSESSMENT

Scenario: Test verifies that the `ReportWriter` creates a directory if it doesn't exist.

Why Needed: This test prevents a bug where the report writer fails to create an output directory when the input JSON file does not exist.

Key Assertions:

- The `tmp_path / 'subdir' / 'report.json'` path should exist after calling `writer.write_report(tests)`.
- The `tmp_path / 'subdir' / 'report.json'` path should be a directory and not an empty file after calling `writer.write_report(tests)`.
- The `tmp_path / 'subdir' / 'report.json'` path should have the correct permissions (e.g. read, write, execute) after calling `writer.write_report(tests)`.
- The `tmp_path / 'subdir' / 'report.json'` path should be a directory with the correct number of subdirectories and files after calling `writer.write_report(tests)`.
- The `tmp_path / 'subdir' / 'report.json'` path should not have any empty directories or files after calling `writer.write_report(tests)`.
- The `tmp_path / 'subdir' / 'report.json'` path should be a directory with the correct permissions (e.g. read, write, execute) when checking its existence using `assert tmp_path / 'subdir' / 'report.json'.exists()`.
- The `tmp_path / 'subdir' / 'report.json'` path should not exist after calling `writer.write_report(tests)` if the input JSON file does not exist.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	84 lines (ranges: 161-165, 167, 169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 229-231, 233, 235, 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510-512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	123 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259,

	262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-477, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_ensure_dir_failure` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the `test_ensure_dir_failure` function to ensure it correctly handles directory creation failures.

Why Needed: This test prevents a potential bug where the report writer fails to capture warnings when creating a non-existent directory.

Key Assertions:

- The function should raise an exception with code 'W201' (Permission denied) when attempting to create the directory.
- The `writer.warnings` list should contain at least one warning with code 'W201'.
- The `writer.warnings` list should not be empty after attempting to create the non-existent directory.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	12 lines (ranges: 156-158, 470-473, 480-484)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_git_info_failure` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the `get_git_info` function handles git command failures gracefully by returning `None` for both SHA and dirty flag when `git not found`,

Why Needed: To prevent a test failure where the test relies on `get_git_info()` to return non-None values, even if it encounters a Git command failure.

Key Assertions:

- The function should return `None` for both `sha` and `dirty` when `git not found` is encountered.
- The function should handle the case where `git not found` raises an exception without raising it again.
- The function should still allow the test to continue running even if `get_git_info()` returns non-None values for other reasons (e.g., successful git command execution).
- The function should provide a clear indication that the git command failed, such as returning `None` or raising an exception with a meaningful error message.
- The function should not silently ignore the failure and return incorrect results.
- The function should handle the case where `git not found` is a known Git version (e.g., older versions) without introducing new bugs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 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` 30ms 🛡️ 5

AI ASSESSMENT

Scenario: Test `test_write_html_creates_file` verifies that the report writer creates an HTML file and includes expected content.

Why Needed: This test prevents a regression where the report writer does not create an HTML file or contains incorrect expected content.

Key Assertions:

- The HTML file should be created at `report.html` in the temporary directory.
- The HTML file should contain the expected tests, including 'test1' and 'test2'.
- The HTML file should display a message indicating that all tests passed ('PASSED') or failed with an error message ('XFailed').
- The HTML file should include messages for skipped tests ('Skipped'), Xfailed tests ('XFailed'), and Xpassed tests ('XPassed').
- The HTML file should contain the expected report header, including 'Test 1', 'Test 2', 'PASSED', 'FAILED', 'Skipped', 'XFailed', and 'XPassed'.

COVERAGE

src/pytest_llm_report/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` 31ms 🛡️ 5

AI ASSESSMENT

Scenario: The test verifies that the report writer includes xfail outcomes in the HTML summary.

Why Needed: This test prevents a regression where xfail outcomes are not included in the HTML summary.

Key Assertions:

- The 'XFAILED' keyword is present in the HTML summary.
- The 'XFailed' keyword is present in the HTML summary.
- The 'XPASSED' keyword is present in the HTML summary.
- The 'XPassed' keyword is present in the HTML summary.
- The 'XFAILED' and 'XFailed' keywords are both present in the HTML summary.
- The 'XPassed' and 'XPassed' keywords are both present in the HTML summary.
- All xfail outcomes are included in the HTML summary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	118 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321, 323-326, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_json_creates_file` 5ms 🛡️ 5

AI ASSESSMENT

Scenario: Test verifies that the `ReportWriter` class successfully creates a JSON file with an associated hash.

Why Needed: This test prevents a bug where the report writer fails to create a JSON file, potentially leading to data loss or inconsistencies.

Key Assertions:

- The `report.json` file should be created in the specified path.
- At least one artifact should be tracked for the report.
- The number of artifacts should be greater than zero.
- The file should exist at the specified path.
- The `ReportWriter` class should successfully create a JSON file with an associated 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/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` 33ms 🛡️ 5

AI ASSESSMENT

Scenario: Test writes PDF file when Playwright is available.

Why Needed: Prevents regression where the test fails due to missing or corrupted report files.

Key Assertions:

- The `report.pdf` path should be created.
- All artifacts in the report directory should have a path equal to the `report.pdf` path.
- The `report.pdf` file should exist at the expected location.
- Any artifacts with paths not matching the `report.pdf` path should be ignored.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 5ms 🛡️ 4

AI ASSESSMENT

Scenario: Test verifies that a warning is raised when missing Playwright is used for PDF output.

Why Needed: This test prevents a regression where the report writer does not warn users when Playwright is missing for PDF output.

Key Assertions:

- The 'report.pdf' file should not exist after the test.
- Any warning code W204_PDF_PLAYWRIGHT_MISSING should be present in the list of warnings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/report_writer.py	98 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230-231, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 401-405, 408)

PASSED `tests/test_report_writer_coverage_v2.py::test_report_writer_ensure_dir_creation` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test ensures directory creation of report files.

Why Needed: Prevents a potential issue where the report file is created without being written to.

Key Assertions:

- The `tmp_dir` exists after the test.
- Any warnings from the report writer are set to 'W202' (directory creation warning).
- The directory containing the report files (`r.html`) is deleted afterwards.

COVERAGE

`src/pytest_llm_report/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` 8ms 🛡️ 5

AI ASSESSMENT

Scenario: Tests the scenario where report_writer_metadata_skips verifies that metadata skips when reports are disabled.

Why Needed: This test prevents regression by ensuring that metadata is skipped when reports are disabled, which helps maintain the expected behavior of the ReportWriter.

Key Assertions:

- The 'start_time' key should be present in the metadata.
- Metadata should not contain the 'llm_model' key.
- The 'llm_model' key should be None.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	36 lines (ranges: 364-380, 382-393, 395, 397, 399, 401, 403, 407, 419)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/report_writer.py`	67 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300)

PASSED `tests/test_schemas.py::TestAnnotationSchema::test_from_dict_full` 1ms 🛡 3

AI ASSESSMENT

Scenario: Test that the `from_dict` method correctly creates an annotation from a dictionary with all required fields.

Why Needed: Prevents regression in handling missing or invalid input data, ensuring consistent behavior across different scenarios.

Key Assertions:

- assert schema.scenario == 'Verify login'
- assert schema.why_needed == 'Catch auth bugs'
- assert schema.key_assertions == ['assert 200', 'assert token']
- assert schema.confidence == 0.95

COVERAGE

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

PASSED `tests/test_schemas.py::TestAnnotationSchema::test_to_dict_full` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Should convert to dictionary with all fields.

Why Needed: Prevent regression in authentication logic by ensuring correct data is passed to the API.

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 79ms 🛡️ 7

AI ASSESSMENT

Scenario: Test that an HTML report is created when the test function `test_simple` is executed.

Why Needed: This test prevents a regression where the HTML report might not be generated correctly if the test function `test_simple` raises an exception.

Key Assertions:

- The file path of the generated report should exist.
- The content of the report should contain the string '
- The string 'test_simple' should be present in the report's content.

COVERAGE

src/pytest_llm_report/collector.py	39 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)

src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	101 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_summary_counts_all_statuses 111ms 🛡️ 7

AI ASSESSMENT

Scenario: tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_summary_counts_all_statuses

Why Needed: This test prevents a regression where the HTML summary counts are not accurate for all statuses.

Key Assertions:

- The 'Total Tests' label should be included in the HTML summary.
- The 'Passed' label should have a count of 1.
- The 'Failed' label should have a count of 1.
- The 'Skipped' label should have a count of 1.
- The 'XFailed' label should have a count of 1.
- The 'XPassed' label should have a count of 1.
- The 'Errors' and 'Error' labels should be included in the HTML summary with correct counts.

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` 65ms 🛡️ 7

AI ASSESSMENT

Scenario: The JSON report is created and contains the expected schema version, summary statistics, and test counts.

Why Needed: This test prevents regression by ensuring that a basic report generation functionality is working correctly.

Key Assertions:

- The `schema_version` key in the report should be set to the current LLM Report JSON format.
- The `summary` key should contain the correct total, passed, and failed counts.
- The `passed` count should match the number of tests that passed.
- The `failed` count should match the number of tests that failed.
- The `report.json` file should exist in the test directory after running the test.
- The JSON data loaded from the report file should contain the expected keys and values.
- The `schema_version` value should be a string representing the current LLM Report JSON format.

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` 75ms 🛡️ 13

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report when a provider is enabled.

Why Needed: Prevent regressions by ensuring LLM annotations are present in the report.

Key Assertions:

- The 'scenario' key in the report should match 'Checks the happy path'.
- The 'why_needed' key in the report should indicate that LLM annotations prevent regressions.
- The 'llm_annotation' key in the report data should contain a 'scenario' value matching 'Checks the happy path'.
- The 'choices' key in the report data should contain at least one 'message' value with a 'content' value matching 'Checks the happy path'.
- The 'key_assertions' list should not be empty.
- The test function 'test_llm_annotations_in_report' should be called before running this test.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	39 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198, 203)
src/pytest_llm_report/llm/base.py	39 lines (ranges: 52-53, 72, 75, 80, 107, 110-111, 128, 136, 147, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218, 245, 247, 249, 252, 257-258, 260)

src/pytest_llm_report/llm/litellm_provider.py	23 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 66-67, 69-70, 94-95, 97)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/models.py	94 lines (ranges: 104-107, 109-111, 113, 115, 161-165, 167, 169, 171, 173, 176, 178-180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407-419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	47 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	186 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-340, 343, 345, 348-352, 355, 357-362, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/prompts.py	29 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-109, 111-112, 116)
src/pytest_llm_report/report_writer.py	105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-296, 298-299, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-

471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_llm_error_is_reported` 6.08s 🛡️ 12

AI ASSESSMENT

Scenario: Verifies that LLM errors are surfaced in HTML output.

Why Needed: Prevents a regression where LLM errors are not reported to the user.

Key Assertions:

- The test verifies that an error message 'LLM error' is present in the report.
- The test verifies that the error message contains the string 'boom'.
- The test checks for the presence of the LLM error message in the HTML output.
- The test ensures that the error message is not empty or null.
- The test verifies that the error message does not contain any other relevant information.
- The test checks if the error message is correctly formatted and readable.

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 53ms 🛡️ 7

AI ASSESSMENT

Scenario: Verify that the LLM opt-out marker is correctly recorded in the report.

Why Needed: This test prevents regression where the LLM opt-out marker is not recorded in the report.

Key Assertions:

- The 'llm_opt_out' marker should be present in the report.
- The 'llm_opt_out' marker should have a value of True for this test.
- There should only be one test with the 'llm_opt_out' marker in the report.

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 53ms 🛡️ 7

AI ASSESSMENT

Scenario: Test the requirement marker to verify it records a specific requirement.

Why Needed: This test prevents a potential bug where multiple requirements are not properly recorded by the requirement marker.

Key Assertions:

- The function `pytester.makepyfile` is called with a string containing the required requirements.
- The `test_with_req` function is defined and passed to `pytester.makepyfile`.
- The test data is loaded from a JSON file using `json.loads`.
- The tests in the report are checked for exactly one requirement.
- The 'requirements' key is present in each test's dictionary.
- The value of 'requirements' is either an empty list or a string containing the required requirements.
- The required requirements are not empty strings.

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 58ms 🛡️ 7

AI ASSESSMENT

Scenario: The test verifies that multiple xfailed tests are recorded in the report.

Why Needed: This test prevents regression by ensuring that all xfailed tests are properly reported and counted.

Key Assertions:

- 2
- ['xfailed', 'xfailed']

COVERAGE

src/pytest_llm_report/collector.py	47 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-116, 119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212, 216, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	74 lines (ranges: 161-165, 167-169, 171, 173, 176, 178, 180, 182, 184, 186, 188, 190, 364-380, 382, 385, 387, 390-393, 395, 397, 399, 401, 403, 407, 419, 449-457, 459, 461, 500, 502-506, 508, 510, 512, 514, 516, 518, 520, 522)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269,

	276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397, 408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/report_writer.py	108 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321, 323-324, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestOutcomes::test_skip_outcome` 54ms 🛡️ 7

AI ASSESSMENT

Scenario: Test that skipped tests are correctly recorded and reported.

Why Needed: This test prevents a regression where the number of skipped tests is not accurately reported in the report.

Key Assertions:

- The total number of skipped tests should be equal to 1 according to the report.
- The 'skipped' key in the report data should contain a single integer value, which represents the total number of skipped tests.
- The 'summary' section of the report data should include a key called 'skipped' with a value of 1.

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` 55ms 🛡️ 7

AI ASSESSMENT

Scenario: Verify that xfailed tests are recorded and reported correctly.

Why Needed: This test prevents a regression where xfailed tests are not properly recorded or reported.

Key Assertions:

- The 'summary' key in the report.json file should contain a count of xfailed tests.
- The 'xfailed' value in the 'summary' key should be equal to 1 (indicating one xfailed test).
- The 'pytester.path' attribute should have been updated with the correct path to the report.json file after running pytest.
- The 'report.json' file should contain a valid JSON object with the expected keys and values.
- The 'summary' key in the 'report.json' file should be present and contain the correct data.
- The 'xfailed' value in the 'summary' key should be an integer (not a string).
- The 'pytester.runpytest' command should have been executed successfully with the '--llm-report-json' option set.

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` 56ms 🛡️ 7

AI ASSESSMENT

Scenario: Test the parameterized tests feature to verify that it records and reports the correct number of tests.

Why Needed: This test prevents a regression where the number of passed tests is not reported correctly in the report.

Key Assertions:

- The total number of tests should be 3 (1 passed, 2 failed).
- Each test should have been run at least once. The 'passed' count should match the actual number of runs for each test.
- No duplicate or missing test counts should be reported in the report.
- All test results should be included in the report even if they are not passed (i.e., all tests ran).
- The 'total' and 'passed' counts should add up to 3 (the total number of tests).

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` 46ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the CLI help text contains usage examples.

Why Needed: This test prevents a potential bug where the help text is missing or misleading.

Key Assertions:

- The result of running `pytester.runpytest('--help')` should contain lines matching `*Example:*--llm-report*` in the output.
- The line containing `*Example:*--llm-report*` should be present in the output.
- The output should not contain any other usage examples besides those mentioned above.
- The output should not contain any examples that are not related to LLM reporting.
- The output should include a clear and concise description of how to use the CLI help text.
- The output should not contain any typos or grammatical errors in the help text.
- The output should be readable and easy to understand, even for users who are not familiar with the command line.

COVERAGE

src/pytest_llm_report/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 41ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that LLM markers are registered and their context is properly set.

Why Needed: To prevent a regression where the LLM marker is not correctly registered or its context is not set.

Key Assertions:

- The 'llm_opt_out' marker should be found in the output.
- The 'llm_context' marker should be found in the output.
- The 'requirement' marker should be found in the output.
- The 'llm_opt_out' marker should not be present in the output if it is properly registered.
- The 'llm_context' marker should only appear once per test case, and its presence should be verified across all tests.
- The 'requirement' marker should only appear once per test case, and its presence should be verified across all 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	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` 47ms 🛡️ 3

AI ASSESSMENT

Scenario: The plugin should be successfully registered with pytest.

Why Needed: This test prevents a potential issue where the plugin is not registered correctly with pytest.

Key Assertions:

- pytester.runpytest('--help')
- result.stdout.fnmatch_lines(['*--llm-report*'])

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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` 79ms 🛡️ 7

AI ASSESSMENT

Scenario: Verify that special characters in nodeid are handled correctly during Pytest execution.

Why Needed: This test prevents a potential crash and ensures the HTML generated by Pytest is valid.

Key Assertions:

- The '<' character should be present in the report.html file.
- The '>' character should not be present in the report.html file.
- The '&' character should not be present in the report.html file.
- The '<' and '>' characters are present in the report.html file, but they are not escaped.
- The HTML content of the report.html file does not contain any special characters.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163-164, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 331, 337-338, 365-375, 387-388, 391, 395-397,

	408, 412, 431, 435-437, 448, 452, 455, 457-458)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	101 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_time.py::TestFormatDuration::test_boundary_one_minute` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the 'format_duration' function with a duration of exactly one minute.

Why Needed: This test prevents a potential bug where the function does not correctly format durations that are close to but less than one minute.

Key Assertions:

- The result should be in the format '1m 0.0s'.
- The time unit 's' should appear after '1' in the result.
- The decimal part of the number should be zero.
- The function should return an error or raise a ValueError for durations less than one minute.
- The function should correctly handle cases where the duration is exactly one minute (e.g., 60.0).
- The function should not incorrectly format other types of durations (e.g., two minutes, three seconds).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

PASSED tests/test_time.py::TestFormatDuration::test_microseconds_format 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 500 microseconds.

Why Needed: This test prevents a potential bug where the function does not correctly format durations as microseconds when they are very close to millisecond values.

Key Assertions:

- The result should contain 'µs' (microseconds).
- The result 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: Verifies that the `format_duration` function correctly formats sub-second durations as milliseconds.

Why Needed: This test prevents a regression where the function incorrectly returns 'ms' when the input is an integer.

Key Assertions:

- The result of `format_duration(0.5)` should be '500.0ms'.
- The value of `result` should match '500.0ms'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	3 lines (ranges: 39, 41-42)

PASSED `tests/test_time.py::TestFormatDuration::test_minutes_format` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the function correctly formats durations over a minute.

Why Needed: This test prevents regression if the format_duration function is modified to incorrectly handle minutes.

Key Assertions:

- The result contains the string 'm' which indicates the unit of measurement (minutes).
- The result equals '1m 30.5s' which correctly formats the duration as one minute and thirty-five seconds.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/time.py`	`6 lines (ranges: 39, 41, 43, 46-48)`

PASSED `tests/test_time.py::TestFormatDuration::test_multiple_minutes` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of multiple minutes.

Why Needed: Prevents regression in formatting durations that include more than one minute.

Key Assertions:

- The result should be '3m 5.0s' when given a duration of 185.0 minutes.
- The result should not have any seconds for durations less than 1 minute.
- The function should correctly handle durations in the format 'mm:ss'.
- The function should preserve the original unit ('minutes') for durations with more than one unit.
- The function should handle negative durations without raising an error.
- The function should support durations of any length (e.g., 2 minutes, 3.5 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/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 `format_duration` function returns a string representation of '1.00s' for a duration of exactly 1 second.

Why Needed: Prevents a potential bug where the test fails due to an incorrect or missing expected result for durations less than 1 second.

Key Assertions:

- The `format_duration` function should return a string representation of '1.00s' when given a duration of exactly 1 second.
- The `format_duration` function should not raise an exception when given a non-numeric input (e.g., a float or integer).
- The `format_duration` function should handle durations less than 1 second correctly and return the expected string representation ('1.00s').

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/util/time.py`	4 lines (ranges: 39, 41, 43-44)

PASSED `tests/test_time.py::TestFormatDuration::test_seconds_format` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the 'seconds' unit of time.

Why Needed: Prevents incorrect formatting of seconds under a minute.

Key Assertions:

- The function should return 's' as part of the result.
- The formatted string should be in the format 'X.Xs'.
- The value of X.X should be exactly 5.50.
- The unit 's' should always be included in the output.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED `tests/test_time.py::TestFormatDuration::test_small_milliseconds` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a small millisecond duration.

Why Needed: This test prevents a potential issue where the function returns incorrect results for very short durations, potentially leading to inaccuracies in time calculations.

Key Assertions:

- The result of calling `format_duration(0.001)` should be '1.0ms'.
- The duration is represented as millisecond (ms) rather than seconds.
- The function handles durations less than one second 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: Verifies that the `format_duration` function correctly formats very small durations as microseconds when the input is 1 microsecond.

Why Needed: This test prevents a potential bug where the `format_duration` function would incorrectly format very small durations as milliseconds instead of microseconds.

Key Assertions:

- The result of calling `format_duration(0.000001)` should be '1µs'.
- The unit of the output value is microsecond (µs).
- The input duration is correctly converted to microseconds.
- No other units are applied when converting 1 microsecond to microseconds.
- The function does not silently truncate or round the result.
- The function handles very small inputs without error.
- The function preserves the original precision of the input value.
- The output value is a string representation of the duration in microseconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	2 lines (ranges: 39-40)

PASSED `tests/test_time.py::TestIsoFormat::test_formats_datetime_with_utc` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test formats datetime with UTC and ensures it is correctly converted to UTC timezone.

Why Needed: This test prevents a potential bug where the `iso_format` function incorrectly converts datetime objects from other timezones to UTC.

Key Assertions:

- The result of `iso_format(dt)` should be in the format `YYYY-MM-DDTHH:MM:SS+HH:MM:SSZ`.
- The timezone offset of the input datetime object should be correctly converted to UTC.
- The resulting string should not have a timezone offset.
- The resulting string should match the expected ISO 8601 format.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 387-388, 391, 395-397)`
`src/pytest_llm_report/util/time.py`	`1 lines (ranges: 27)`

PASSED `tests/test_time.py::TestIsoFormat::test_formats_naive_datetime` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the naive datetime format without timezone to ensure it matches the expected output.

Why Needed: This test prevents a potential bug where the naive datetime format is not correctly converted to UTC time zone.

Key Assertions:

- The function `iso_format(dt)` returns the correct ISO formatted string for the given datetime object.
- The datetime object `dt` is in the naive timezone (i.e., no timezone offset).
- The resulting ISO formatted string matches the expected output '2024-06-20T14:00:00'.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/time.py`	1 lines (ranges: 27)

PASSED `tests/test_time.py::TestIsoFormat::test_formats_with_microseconds` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `iso_format` function with a datetime object that includes microseconds.

Why Needed: This test prevents a potential issue where the `iso_format` function does not correctly handle datetime objects with microseconds.

Key Assertions:

- The result of calling `iso_format(dt)` should contain the string '123456'.
- The value in the result should be exactly '123456'.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/time.py`	1 lines (ranges: 27)

PASSED `tests/test_time.py::TestUtcNow::test_has_utc_timezone` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `utc_now()` function returns a datetime object with an associated UTC timezone.

Why Needed: Prevents regression in time-related functionality when working with dates and times.

Key Assertions:

- The returned datetime object has a valid timezone.
- The returned datetime object is not None (i.e., it's a valid result).
- The returned datetime object's timezone is equal to UTC.
- The returned datetime object does not have an invalid or missing timezone.
- The returned datetime object's timezone is correctly set to UTC.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 387-388, 391, 395-397)
`src/pytest_llm_report/util/time.py`	1 lines (ranges: 15)

PASSED `tests/test_time.py::TestUtcNow::test_is_current_time` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `utc_now()` function returns a current time within UTC.

Why Needed: This test prevents a potential issue where the `utc_now()` function returns an incorrect or outdated time due to clock skew.

Key Assertions:

- The result of `utc_now()` should be within one second of the current UTC time.
- The result of `utc_now()` should not exceed the current UTC time by more than one second.
- The result of `utc_now()` should not be less than the current UTC time by more than one second.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 387-388, 391, 395-397)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)

PASSED `tests/test_time.py::TestUtcNow::test_returns_datetime` 1ms 🛡 3

AI ASSESSMENT

Scenario: The `utc_now()` function returns a datetime object.

Why Needed: This test prevents a potential issue where the returned datetime object might not be in UTC.

Key Assertions:

- result is an instance of `datetime`
- result is not None
- result is of type `datetime` (not `timedelta`)
- result is not a timezone-aware datetime object (it's naive)
- result has a valid year, month and day
- result has a valid hour, minute and second
- result has the correct timezone

COVERAGE

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