Test Report

Run ID: 21118757847-py3.12 • Generated: 2026-01-18 21:19:09 • Duration: 33.18s
Plugin: v0.2.0 (b7a157f6cb9189cc50a17c846484c8454deeac61) [dirty]
Repo: v0.2.0 (64516b02bcb80857186322923 9cfd8a3ef690f50)
LLM: ollama / llama3.2:1b (minimal context, 486 annotated, 45 errors)

92.96%
Total Coverage

532	532	0	0
TOTAL TESTS	PASSED	FAILED	SKIPPED

0	0	0
XFAILED	XPASSED	ERRORS

Source Coverage Per Test Details Failures Only

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	117	5	112	95.73%	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, 147, 149, 151, 165, 167, 171, 173, 175, 185, 187-191, 193-194, 197, 199, 208, 220, 222-236, 238, 240, 248-249, 251-252, 254, 256-258, 262, 265-266, 268-269, 272,	66, 90-91, 195, 206

					274-275, 277, 279-280, 284	
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	6	129	95.56%	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, 127-128, 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,	62, 123, 125, 157, 221, 251

					216-217, 220, 224-225, 228-234, 236, 239, 241, 243-244, 246-250, 252-254, 257, 259-260, 263-264, 271, 273-274, 276-279, 281-283, 285, 299-300, 302, 308	
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	278	23	255	91.73%	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, 225-226, 235, 237-238, 242-243, 246-247, 249-250, 258-259, 266, 272-273, 275, 279-283, 285-289, 292-293, 298-299, 306-307, 309, 321, 323-324, 328, 333, 336-338, 341-349, 351-352, 354, 358-361, 363, 366-372, 374-380, 386-388, 390-393, 395, 397-398, 402-408, 411, 414-416, 418-420, 422-427, 433-434, 436-440, 443-446, 448-449, 451-453	89, 104, 106, 115-117, 199, 228-229, 233, 239-241, 248, 251-254, 256, 262, 373, 447, 450
src/pytest_llm_report/llm/litellm_provider.py	62	4	58	93.55%	8, 10, 12-13, 21, 31, 37-38, 41-42, 44, 51, 60-62, 64, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 108, 110, 114, 116, 118-120, 129, 131, 133, 135-136, 138, 142, 164, 175-176, 179-181, 183, 185-186,	123-124, 126, 191

					188, 190, 195, 197, 199, 205-206, 208	
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	45	2	43	95.56%	7, 9, 11-12, 18, 24, 40-41, 47, 50, 52, 54-55, 57-60, 62, 64-65, 71, 73, 76-77, 79-80, 82, 86, 92-93, 95-97, 101, 107, 109, 119, 121-122, 132, 137, 139-140	69, 75
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/llm/token_refresh.py	71	0	71	100.0%	7, 9-14, 17, 20, 23-24, 36-39, 41-43, 47, 59-60, 63-66, 69-72, 74, 83, 85-88, 90-91, 93, 101-103, 107-109, 111, 113-116, 120, 132-136, 139-140, 143-145, 148-150, 153-156, 158, 160-162	-
src/pytest_llm_report/models.py	243	0	243	100.0%	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, 159-160, 162, 164, 166, 173-196, 199-200, 208-209, 211, 213, 219-220, 229-231, 233, 235, 239-241, 244-245, 254-256, 258, 260, 267-268, 277-279, 281, 283, 287-	-

					289, 292-293, 330-359, 361-366, 368, 370, 388-411, 413-425, 428-429, 443-451, 453, 455, 465-467, 470-471, 488-498, 500, 506, 508, 514-528	
src/pytest_llm_report/options.py	197	51	146	74.11%	122, 162, 191, 194-196, 201-203, 209-211, 217-219, 225-227, 233-234, 237-246, 248, 252, 261, 276, 279-280, 288-293, 295, 300-305, 308-313, 316-317, 320-321, 324-325, 328-337, 340-343, 346-359, 362-363, 366-367, 370-375, 380-381, 384-385, 390-393, 398-401, 403, 405, 407, 414-415, 417-418, 420-421, 424-437, 440-449, 452, 454	13-15, 21-22, 98-102, 105-107, 110-115, 118-121, 138-139, 142-148, 151-153, 156-158, 161, 172-176, 179-180, 183, 185, 250, 255, 264
src/pytest_llm_report/plugin.py	147	24	123	83.67%	40, 43, 49, 55, 61, 67, 73, 80, 89, 95, 101, 107, 113, 121, 127, 133, 150, 154, 158, 164-165, 168-169, 171, 173, 176-178, 184-185, 193-194, 221-222, 225-226, 229, 232-233, 235-236, 239-240, 242, 244-248, 251-252, 254, 256, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-288, 290, 292-295, 298-299, 307-308, 313-316, 319, 321, 324-329, 331, 333, 341-342, 363-364, 367-368, 371-373, 384-385, 388, 391-392, 395-397,	13, 15-17, 19-20, 22, 28-31, 34, 141, 199, 303-304, 309-310, 355-356, 376, 400, 416-417

					407-408, 411-413, 424-425, 428, 431, 433-434	
src/pytest_llm_report/prompts.py	75	1	74	98.67%	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, 114, 116, 118, 132-133, 135-138, 140-142, 144-146, 148-149, 151-152, 154-156, 158-159, 163, 165, 180, 182, 191-194	80
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	3	164	98.2%	13, 15-25, 27-29, 46, 55, 58, 67-68, 76, 83-84, 89, 98-100, 102, 105-108, 110, 113, 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, 414-425, 432,	135-137

					434-435, 437-439, 447-451, 453, 455, 458, 461-462, 464, 470-474, 480-481, 488, 495, 497, 499-501, 503, 506-507, 509, 515-516	
src/pytest_llm_report/util/fs.py	34	1	33	97.06%	11, 13-14, 17, 30, 33, 36, 39, 42, 45, 55-56, 58-60, 63-65, 67, 70, 79, 82, 100, 103, 111-113, 116-117, 119-121, 123	40
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	-

Per Test Details

📄 tests/test_aggregation.py

10 tests

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

AI ASSESSMENT

Scenario: Test `test_aggregate_all_policy` verifies that the aggregate function correctly handles multiple reports from different nodes and phases.

Why Needed: This test prevents a regression where the aggregate function might incorrectly retain tests for non-existent reports or failed tests.

Key Assertions:

- The aggregated report should contain both retained tests.
- The length of the aggregated report's `tests` list should be equal to 2.
- Both retained tests should exist in the aggregated report.
- Failed tests should not be included in the aggregated report.
- Non-existent reports should not be included in the aggregated report.
- All tests from both nodes and phases should be included in the aggregated report.

COVERAGE

src/pytest_llm_report/aggregation.py	70 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, 147, 149, 151-156, 158, 160-162, 173, 220, 222-226, 238, 248, 251-252, 254, 256, 279-282, 284)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the aggregate function fails to aggregate data in cases where the aggregation directory does not exist.

Key Assertions:

- The `aggregator.aggregate()` method should return `None` when the specified `aggregate_dir` is not found or does not exist.
- The `pathlib.Path.exists()` mock returns `False` for the specified `aggregate_dir`.
- No exception is raised when the aggregation directory does not exist, preventing potential data loss.
- The aggregate function behaves as expected in this scenario, indicating a bug-free implementation.
- The test fails with an assertion error if the `aggregator.aggregate()` method returns a different value than `None` for a non-existent `aggregate_dir`.
- The `pathlib.Path.exists()` mock ensures that the `exists` method is called only when necessary, preventing unnecessary computations.
- The test case covers all possible scenarios where the aggregation directory does not exist or is not found, ensuring robustness and reliability.

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: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where the `aggregate` function would incorrectly pick an older policy when multiple tests are run on different times.

Key Assertions:

- The `outcome` of the aggregated report should be 'passed' if it is the latest policy for the given test case.
- The number of tests in the aggregated report should be 1.
- The outcome of the first test in the aggregated report should be 'passed'.
- The `run_meta.is_aggregated` flag should be True.
- The `run_meta.run_count` should match the number of runs (2) on different times.
- The summary of the aggregated report should have exactly 1 passed test and 0 failed tests.

COVERAGE

src/pytest_llm_report/aggregation.py	78 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, 147, 149, 151-156, 158, 160-162, 173, 185, 187-191, 193-194, 197, 220, 222-226, 238, 248, 251-252, 254, 256, 279-282, 284)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

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

Key Assertions:

- agg.aggregate() should return None
- mock_config.aggregate_dir is set to None before calling aggregate()
- No exception is thrown when agg.aggregate() is called

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: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that `aggregate` returns `None` when there are no reports.

Why Needed: Prevents regression in case of an empty report directory.

Key Assertions:

- The function should return `None` without attempting to aggregate any data.
- No reports should be found using the specified path.
- There should not be any aggregated data returned.
- The `aggregate` method should not raise an exception when called with no arguments.
- The `aggregate` method should not modify the original directory.
- The `aggregate` method should not return a report.
- No error should be raised if the directory is empty.
- The function should handle cases where the directory does not exist.

COVERAGE

src/pytest_llm_report/aggregation.py	9 lines (ranges: 52, 55-57, 109-110, 113-114, 173)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

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

Key Assertions:

- Coverage was properly deserialized with correct file paths and line ranges.
- LLM annotation was properly deserialized with correct scenario, why needed, and key assertions.
- Can be re-serialized without any issues (this would fail before the fix).

COVERAGE

src/pytest_llm_report/aggregation.py	82 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, 147, 149, 151-156, 158, 160-162, 173, 185, 187-191, 197, 220, 222-226, 238, 248, 251-252, 254, 256, 279-282, 284)
src/pytest_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	34 lines (ranges: 40-43, 104-107, 109-111, 113, 115, 162, 166-171, 173, 175, 177, 179, 182-186, 188, 190, 192, 194, 196)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `aggregate` method returns a single `SourceCoverageEntry` for each source file.

Why Needed: This test prevents regression where multiple source files are aggregated into one entry.

Key Assertions:

- The `source_coverage` list contains exactly one element with the expected structure.
- Each `SourceCoverageEntry` has a `file_path` attribute matching the expected value.
- All `SourceCoverageEntry` attributes match the expected values.
- The `coverage_percent` and `covered_ranges`/`missed_ranges` attributes are calculated correctly.

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, 151-158, 160-162, 173, 185, 187-189, 197, 220, 222-223, 238, 248, 251-252, 254, 256, 279-282, 284)
`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a bug where the aggregator fails to load coverage data when the `llm_coverage_source` option is not provided or the specified file does not exist.

Key Assertions:

- Verify that `_load_coverage_from_source()` returns `None` when `llm_coverage_source` is set to `None`.
- Verify that `_load_coverage_from_source()` raises a `UserWarning` with the message 'Coverage source not found' when the specified file does not exist.
- Verify that the mock coverage object returned by `_load_coverage_from_source()` has the correct attributes and methods when successfully loaded from a `.coverage` file.
- Verify that the mock mapper object returned by `_load_coverage_from_source()` has the correct `map_source_coverage` method call when the specified file does not exist.
- Verify that the coverage percentage is correctly reported as 80.0 when successfully loaded from a `.coverage` file.

COVERAGE

src/pytest_llm_report/aggregation.py	19 lines (ranges: 248-249, 251-252, 254, 256-260, 262, 265-266, 268-269, 272, 274-275, 277)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `_recalculate_summary` method correctly recalculates the summary when new test results are added.

Why Needed: This test prevents regression where a new test result would cause the summary to be outdated.

Key Assertions:

- The total count of tests should match the provided number.
- The passed count should match the provided number.
- The failed count should match the provided number.
- The skipped count should match the provided number.
- The xfailed count should match the provided number.
- The xpassed count should match the provided number.
- The error count should match the provided number.
- The coverage percentage should be preserved.
- The total duration of all tests should remain unchanged.

COVERAGE

`src/pytest_llm_report/aggregation.py`	`17 lines (ranges: 220, 222-236, 238)`
`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

PASSED tests/test_aggregation.py::TestAggregator::test_skips_invalid_json 3ms 3

AI ASSESSMENT

Scenario: Test verifies that skipping an invalid JSON report prevents a regression.

Why Needed: This test ensures that the aggregation function correctly handles reports with missing or invalid fields.

Key Assertions:

- The 'aggregate' function should not count any report when it encounters an invalid JSON file.
- The 'aggregate' function should raise a warning for skipping an invalid report file.
- The 'aggregate' function should only count the valid report in its run meta.
- The 'aggregate' function should ignore reports with missing fields.
- The 'aggregate' function should not include any skipped reports in its final 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, 151-156, 158, 160-162, 165, 167-169, 171, 173, 185, 187-189, 197, 220, 222-223, 238, 248, 251-252, 254, 256, 279-282, 284)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_aggregation_maximal.py

1 tests

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 outcomes and durations.

Why Needed: This test prevents regression in the aggregator's behavior when dealing with incomplete or inconsistent data.

Key Assertions:

- The total duration of the summary should be equal to the sum of the individual test durations.
- The number of passed tests should match the expected outcome (1 passed, 1 failed).
- The coverage total percent should remain unchanged after recalculating the summary.
- The total duration of the summary should be greater than or equal to the longest test duration.
- The coverage total percent should still be within the specified range (88.5%).

COVERAGE

`src/pytest_llm_report/aggregation.py`	`10 lines (ranges: 44, 220, 222-228, 238)`
`src/pytest_llm_report/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: 123, 163)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

📄 tests/test_annotator.py

7 tests

PASSED `tests/test_annotator.py::TestAnnotateTests::test_cached_tests_are_skipped` 1ms 5

AI ASSESSMENT

Scenario: Verify that cached tests are skipped for the annotator test.

Why Needed: This test prevents a regression where the annotator test is run multiple times with the same input data and the results are cached.

Key Assertions:

- Mocking `mock_provider` to return an empty list or None when called.
- Mocking `mock_cache` to not store any results for the given input data.
- Mocking `mock_assembler` to not generate any output for the given input data.
- Verifying that the test does not run multiple times with the same input data and cached results.
- Checking that the test only runs once with unique input data and no cached results.
- Verifying that the cache is properly cleared after each test run.

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The annotator function is called concurrently with multiple providers and caches.

Why Needed: This test prevents a potential performance regression where the annotator function may be called multiple times in quick succession without being properly synchronized.

Key Assertions:

- mock_provider.assert_called_once_with('provider1')
- mock_provider.assert_called_once_with('provider2')
- mock_cache.assert_called_once_with('cache1', 'annotation1')
- mock_assembler.assert_called_once_with('assembly1', 'annotation1')

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that concurrent annotation handling prevents failures.

Why Needed: This test ensures that annotators can handle failures in a concurrent manner without crashing or producing unexpected results.

Key Assertions:

- Mocking the annotator, cache, and assembler objects to prevent any exceptions from being raised.
- Using capsys to capture the output of the annotation process.
- Verifying that the annotation process completes successfully even when there are failures.
- Checking for any errors or warnings produced by the annotation process.
- Ensuring that the annotator can handle failures without crashing or producing unexpected results.
- Testing that the cache is not affected by concurrent annotation handling failures.
- Verifying that the assembler does not produce any output when an error occurs during 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	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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the progress reporting is implemented correctly in the test.

Why Needed: The current implementation may not accurately track progress or provide meaningful insights for annotators.

Key Assertions:

- mock_provider.progress_reporting() should return a mock object with the expected attributes
- mock_cache.progress_reporting() should be called before and after each annotation
- mock_assembler.progress_reporting() should be called before and after each annotation
- mock_provider.progress_reporting().get_progress() should return a valid progress value
- mock_cache.progress_reporting().get_progress() should return a valid progress value
- mock_assembler.progress_reporting().get_progress() should return a valid progress value

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verifies the sequential annotation of a test function.

Why Needed: Prevents a potential regression in the annotator's behavior when multiple tests are run sequentially.

Key Assertions:

- The `test_sequential_annotation` method should not throw an exception when called with multiple test functions.
- The `mock_provider`, `mock_cache`, and `mock_assembler` mocks should be created without any side effects.
- The `test_sequential_annotation` method should return a mock object that is not equal to the original function.
- The `mock_provider` mock should have been called with the correct arguments.
- The `mock_cache` mock should have been called with the correct arguments.
- The `mock_assembler` mock should have been called with the correct arguments.
- The `test_sequential_annotation` method should not raise an exception when called multiple times.

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verifies the annotator function behaves as expected when LLM is disabled.

Why Needed: Prevents a potential bug where the annotator function does not skip tests if LLM is disabled.

Key Assertions:

- The `Config` object with 'none' provider is created.
- An empty list of annotations is passed to the `annotate_tests` function.
- No annotation results are returned from the `annotate_tests` function.
- The annotator function does not skip any tests when LLM is disabled.
- A test case should be skipped if LLM is disabled.
- The `skip` method of the `Config` object with 'none' provider should return False.
- An error message or exception should be raised when trying to annotate tests with an empty list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	2 lines (ranges: 45-46)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The annotator should skip annotation if the provider is unavailable.

Why Needed: This test prevents a potential regression where the annotator may continue to annotate without checking for provider availability.

Key Assertions:

- Mocked provider is not available
- Annotation skipped due to provider unavailability
- No annotation performed when provider is unavailable

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_annotator_maximal.py

4 tests

PASSED `tests/test_annotator_maximal.py::TestAnnotatorAdvanced::test_annotate_concurrent_with_progress_and_errors` 3ms 4

AI ASSESSMENT

Scenario:
tests/test_annotator_maximal.py::TestAnnotatorAdvanced::test_annotate_concurrent_with_progres

Why Needed: Prevents a potential bug where the annotator does not report errors and progress messages concurrently, potentially leading to missing information about the annotation process.

Key Assertions:

- The function should append 'first error' to the `progress_msgs` list when an error occurs.
- The function should check if any of the `progress_msgs` contain 'LLM annotation'.
- The function should report progress messages containing 'Processing X test(s)' for at least 2 tasks.
- The function should not return immediately after annotating all tasks without reporting errors or progress messages.
- The function should handle cases where there are no annotations (i.e., `failures == 0`).
- The function should report the first error found in the annotation process.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: test_annotate_sequential_rate_limit_wait

Why Needed: Prevents regression when rate limit intervals are not yet reached.

Key Assertions:

- The time.sleep() function was called before the annotation task completed.
- The time.sleep() function was called after the annotation task completed.
- The time.sleep() function was called with a delay of less than or equal to 1.0s.
- The time.sleep() function was called with a delay greater than 1.0s but less than or equal to 2.0s.
- The time.sleep() function was called with a delay greater than 2.0s.
- The time.sleep() function did not call before the annotation task completed.
- The time.sleep() function did not call after the annotation task completed.

COVERAGE

src/pytest_llm_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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Should report progress for cached tests.

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

Key Assertions:

- The `get_provider` method of `LLMCache` should return a mock provider instance.
- The `assemble` method of the `ContextAssembler` class should be called with a tuple containing 'src' and None.
- Any progress messages in the `progress_msgs` list should contain '(cache): test_cached'.
- The `is_available` method of `mock_provider` should return True.
- The `get` method of `mock_cache` should return a mock annotation.
- The `append` method of `progress_msgs` should append a string containing '(cache): test_cached'.
- Any progress messages in the `progress_msgs` list should not contain any other strings.

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the annotator returns a message indicating that the provider is not available when it is not available.

Why Needed: This test prevents regression where the annotator fails to report an error when the provider is unavailable, potentially masking bugs or incorrect behavior.

Key Assertions:

- mock_provider.is_available.return_value == False
- assert captured.out.startswith('not available. Skipping annotations')
- mock_provider.get_provider() != mock_provider
- assert mock_provider.is_available.called_once_with() == True
- assert mock_provider.is_available.call_count == 1
- assert mock_provider.return_value.message == 'not available. Skipping annotations'
- assert mock_provider.return_value.status_code == 500

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_base_coverage_v2.py

2 tests

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the function does not handle non-string fields in its response data.

Key Assertions:

- assert annotation.scenario == '123'
- assert annotation.why_needed == ['list']
- assert annotation.key_assertions == ['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: 123, 163)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

📄 tests/test_base_maximal.py

9 tests

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

AI ASSESSMENT

Scenario: Verify that the `get_gemini_provider` function returns a `GeminiProvider` instance.

Why Needed: This test prevents a potential bug where the `get_gemini_provider` function might return an incorrect provider type if the configuration is invalid or missing.

Key Assertions:

- The function `get_provider(config)` should return an instance of `GeminiProvider`.
- The returned `GeminiProvider` instance should have a valid `config` attribute.
- The `GeminiProvider` instance should be able to provide a `gemini_url` attribute.
- The `gemini_url` attribute of the `GeminiProvider` instance should be a string.
- The `get_provider(config)` function should not raise an exception if the configuration is invalid or missing.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents the introduction of unknown LLM providers in the future.

Key Assertions:

- The function `get_provider` raises a `ValueError` with the message 'Unknown LLM provider: invalid'.
- The error message includes the string 'invalid' which is the name of the unknown provider.
- The test verifies that the error message contains the exact phrase '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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: Prevents regression in case of an incorrect provider configuration.

Key Assertions:

- The returned provider is indeed an instance of `LiteLLMProvider`.
- The provider is not None, indicating it was successfully retrieved.
- The provider has the correct type (LitELLMProvider).
- No exceptions were raised during the retrieval process.
- The provider's attributes match the expected values (e.g., name, description).

COVERAGE

src/pytest_llm_report/collector.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/llm/litellm_provider.py	3 lines (ranges: 37-38, 41)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that a NoopProvider instance is created when the 'provider' parameter is set to 'none'.

Why Needed: This test prevents a potential bug where a NoopProvider instance is not created if the 'provider' parameter is set to 'none'.

Key Assertions:

- The `get_provider` function returns an instance of `NoopProvider` when the 'provider' parameter is set to 'none'.
- The `config` object passed to `get_provider` has a valid `provider` attribute.
- The `provider` attribute of the returned `NoopProvider` instance is not None.
- A NoopProvider instance is created with no provider information.
- The `provider` method of the `NoopProvider` instance does not raise an exception when called.
- The `get_provider` function correctly handles a 'none' provider value.
- The test fails if the 'provider' parameter is set to anything other than 'none'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 245, 247, 249-250)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `OllamaProvider` is not correctly instantiated from the provided configuration.

Key Assertions:

- The `provider` parameter passed to `get_provider(config)` should be an instance of `ollama.OllamaProvider`.
- The returned `provider` should have the correct type and attributes.
- The `provider` attribute of the returned `provider` should be set correctly.
- The `OllamaProvider` class should be instantiated correctly from the provided configuration.
- The `get_provider` function should return an instance of `ollama.OllamaProvider` when given a valid configuration.
- The `get_provider` function should raise an error if it cannot instantiate the `OllamaProvider` class.
- The `provider` attribute of the returned `provider` should be set correctly after instantiation.

COVERAGE

src/pytest_llm_report/collector.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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the LlmProvider defaults implement _check_availability correctly.

Why Needed: The test prevents a potential regression where the default LlmProvider does not implement _check_availability.

Key Assertions:

- The provider should be available.
- The provider should have one check.
- The checks counter should increment each time _check_availability 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/llm/base.py	6 lines (ranges: 52-53, 107-108, 110-111)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The `get_model_name()` method of the `ConcreteProvider` class should return the default model name specified in the configuration when no custom model is provided.

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

Key Assertions:

- The `get_model_name()` method of the `ConcreteProvider` class should return 'test-model'.
- The `get_model_name()` method of the `ConcreteProvider` class should not raise an exception when no custom model is provided.
- The default model name specified in the configuration should be 'test-model' if it exists, otherwise it should be a default value like 'default_model'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 136)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 concrete LLM provider returns `None` when no default rate limits are specified.

Why Needed: This test prevents a potential bug where the default rate limits for an LLM provider are not properly set, leading to unexpected behavior or errors in downstream applications.

Key Assertions:

- The method `get_rate_limits()` is called on the concrete LLM provider.
- The result of calling `get_rate_limits()` is `None`.
- The absence of default rate limits is indicated by a value of `None` for this property.
- A concrete LLM provider instance is created with no specified default rate limits.
- The `get_rate_limits()` method is called on the non-defaulted LLM provider instance.
- The result of calling `get_rate_limits()` is still `None` even after a default rate limit has been set.
- A default rate limit is explicitly set for the concrete LLM provider instance before calling `get_rate_limits()`.
- The absence of a specified default rate limit indicates that this property should be set to `None`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 128)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that `is_local()` returns False for a non-local configuration.

Why Needed: Prevents regression in case the default configuration is not local.

Key Assertions:

- The function `provider.is_local()` should return `False` for a valid configuration.
- A valid configuration should have a `local` key set to `True` or `False`.
- If `local` is not present, the function should raise an exception or return an error message.
- The value of `local` in the configuration dictionary should be either `True` or `False`.
- A non-local configuration should have a `local` key with a value that indicates it's not local.
- If the default configuration is not local, the function should raise an exception or return an error message.
- The function should handle cases where `local` is set to `None` in the configuration 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/llm/base.py	3 lines (ranges: 52-53, 147)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_cache.py

7 tests

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

AI ASSESSMENT

Scenario: The 'hash_source' function is called with the same source code, and it returns the expected hash value.

Why Needed: This test prevents a regression where different inputs to the 'hash_source' function produce different hashes due to caching issues.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

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: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the hash length is not as expected due to caching or other external factors.

Key Assertions:

- The hash should be exactly 16 characters long.
- The hash should not be shorter than 16 characters.
- The hash should not be longer than 16 characters.
- The hash generated by `hash_source` is a string of only hexadecimal digits.
- The hash generated by `hash_source` has leading zeros (e.g. '0000...')
- The hash generated by `hash_source` does not contain any non-hexadecimal characters.
- The hash generated by `hash_source` meets the expected length and content requirements.

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: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test clearing the LLM cache and verifying that it removes all entries.

Why Needed: This test prevents a bug where the cache is not cleared correctly after adding some entries, causing incorrect assertions to be made about its contents.

Key Assertions:

- Verify that the `clear` method returns the correct number of cache entries (2 in this case).
- Verify that the `get` method for a specific key returns `None` when called with that key after clearing the cache.
- Verify that all cache entries are removed from the cache after clearing it.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that annotations with errors are not cached.

Why Needed: Prevents a potential regression where an error in the annotation causes it to be cached and then not removed even after the error has been resolved.

Key Assertions:

- The cache should not contain the annotation for 'test::foo' with the key 'abc123' when the result of the get operation is None.
- The cache should remove the annotation for 'test::foo' with the key 'abc123' after the error has been resolved.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test 'get_missing' function of LlmCache class.

Why Needed: This test prevents a potential bug where the function returns `None` when trying to retrieve a non-existent key from the cache.

Key Assertions:

- The function should return None for a missing entry in the cache.
- The function should not raise any exceptions for missing entries.
- The function should handle the case of an empty cache correctly.
- The function should not throw an error when trying to retrieve a non-existent key.
- The function should be able to handle different types of keys (e.g., strings, integers).
- The function should be able to handle missing entries with different values (e.g., None, empty string).

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 LLMCache stores annotations in a secure manner.

Key Assertions:

- Check that the annotation is set correctly with the given key and value.
- Check that the annotation's confidence level matches the expected value.
- Verify that the retrieved annotation has the same scenario, why_needed, and confidence as the original annotation.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_collector.py

11 tests

PASSED `tests/test_collector.py::TestCollectorCollectionErrors::test_collection_error_structure` 1ms 2

AI ASSESSMENT

Scenario: The test verifies that a `CollectionError` object has the correct 'nodeid' and 'message' attributes.

Why Needed: This test prevents a potential bug where a collection error is not properly structured, potentially leading to incorrect handling or reporting of errors in the code.

Key Assertions:

- The 'nodeid' attribute should be set to the value provided in the error message.
- The 'message' attribute should contain the correct error message.
- The 'nodeid' and 'message' attributes should match the expected values.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

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: Prevents a potential bug where an empty collection is returned unexpectedly, potentially causing issues downstream in the test suite.

Key Assertions:

- The `get_collection_errors` method returns an empty list when called on an empty configuration.
- A non-empty collection is not returned when called on an empty configuration.
- The `get_collection_errors` method raises a `ValueError` exception when called with an empty configuration.
- An error message is provided by the `get_collection_errors` method when called with an empty configuration.
- The `get_collection_errors` method does not raise any exceptions when called with an empty configuration.
- The `get_collection_errors` method returns a default value (None) when called with an empty configuration.
- A different error message is provided by the `get_collection_errors` method when called with an empty configuration.
- The `get_collection_errors` method raises a `TypeError` exception when called with an empty configuration.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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.

Key Assertions:

- The llm_context_override attribute should be None for the given TestCaseResult.
- The llm_context_override attribute does not contain any string values.
- The llm_context_override attribute does not contain any boolean values.
- The llm_context_override attribute is set to a value other than None or an empty string.
- The llm_context_override attribute has the correct type (str).
- The llm_context_override attribute is not None for all TestCaseResult instances.
- The llm_context_override attribute is None for all TestCaseResult instances with a given nodeid and outcome.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 when not specified.

Why Needed: Prevents a bug where the default value of llm_opt_out is incorrectly set to True, potentially causing incorrect results in downstream tests or reports.

Key Assertions:

- The llm_opt_out attribute is correctly initialized with False.
- The llm_opt_out attribute does not have any dependencies that could cause it to be set to True.
- The TestCaseResult class has a correct implementation of the llm_opt_out attribute.
- The test passes without any errors or warnings when running with default configuration.
- The test fails with an error or warning when running with specific configuration that sets llm_opt_out to True.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the output capture feature is accidentally enabled by default.

Key Assertions:

- config.capture_failed_output should be set to False
- output_capture_enabled should be False
- no_output_capture_message should be displayed
- capture_output_path should not be provided
- log_level should not be set to DEBUG for output capture
- test_output_capture_enabled 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/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The TestCollectorOutputCapture test verifies that the default value of `capture_output_max_chars` is correctly set to 4000.

Why Needed: This test prevents a potential bug where the default max chars is not set to 4000, potentially leading to unexpected behavior or errors in the application.

Key Assertions:

- The `capture_output_max_chars` attribute of the TestCollectorOutputCapture class should be equal to 4000.
- The `capture_output_max_chars` attribute does not exceed 4000.
- The default value of `capture_output_max_chars` is correctly set to 4000.
- Increasing or decreasing the value of `capture_output_max_chars` will not affect the test outcome.
- The TestCollectorOutputCapture class is properly initialized with a valid configuration.
- The `capture_output_max_chars` attribute is updated correctly after initialization.

COVERAGE

src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test 'xfail failures should be recorded as xfailed' is not being tested.

Why Needed: This test prevents regression where the collector incorrectly records failed tests as expected failure.

Key Assertions:

- The `results` dictionary contains an entry for `test_xfail.py::test_expected_fail` with a value of 'xfailed'.
- The `outcome` attribute is set to 'xfailed' in the `results` dictionary.
- The `wasxfail` attribute is set to 'expected failure' in the `results` dictionary.
- The `duration` and `longrepr` attributes are not relevant to this test, but they may be useful for debugging purposes.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: xfail passes should be recorded as xpassed.

Why Needed: To prevent regression in case of unexpected failures during test runs.

Key Assertions:

- The `result.outcome` property is set to 'xpassed' when the report indicates an expected failure.
- The `collector.results[report.nodeid]` attribute returns a dictionary with the correct key ('xpassed') for the given node ID.
- The test verifies that the collector correctly marks xfail tests as passed after handling their log reports.
- The test ensures that the collector's results are consistent with the expected outcome of an unexpected failure.
- The test checks that the `result.outcome` property is set to 'xpassed' even when the report indicates a failed test.
- The test verifies that the collector correctly handles xfail tests and marks them as passed after handling their log reports.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `create_collector` method initializes with an empty results dictionary, potentially causing unexpected behavior or errors in subsequent operations.

Key Assertions:

- The `results` attribute of the `collector` object is set to an empty dictionary when created.
- The `collection_errors` list is empty when the `collector` object is created.
- The `collected_count` attribute is set to 0 when the `collector` object is created.
- The `results` attribute is not updated with any data when a new collection is added using `add()` method.
- The `collection_errors` list is not updated with any errors when an error occurs during collection.
- The `collected_count` attribute is not incremented correctly when the same collector is used multiple times.
- The `results` attribute is not checked for changes after a new collection is added using `add()` method.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where the order of results is not maintained due to manual additions or modifications.

Key Assertions:

- The list of node IDs returned by `get_results()` should be in ascending order based on their values.
- The list of node IDs returned by `get_results()` should contain only unique node IDs.
- The sorted list of node IDs should maintain the original order of results before any manual additions or modifications.
- No duplicate node IDs should be present in the sorted list.
- The first element of the sorted list should be 'a_test.py::test_a'.
- The second element of the sorted list should be 'z_test.py::test_z'.

COVERAGE

src/pytest_llm_report/collector.py	15 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 bug where the count of collected items is not updated correctly after the collection finish event.

Key Assertions:

- The `collected_count` attribute should be set to 3 (number of collected items)
- The `deselected_count` attribute should be set to 1 (number of deselected items)

COVERAGE

`src/pytest_llm_report/collector.py`	`16 lines (ranges: 78-79, 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 123, 163)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

📄 tests/test_collector_maximal.py

14 tests

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

AI ASSESSMENT

Scenario: Test should not capture output if config disabled (integration via handle_runtest_logreport).

Why Needed: To prevent capturing of output when the `capture_failed_output` configuration flag is set to False.

Key Assertions:

- The `collector.handle_runtest_logreport(report)` call should not modify the report's captured stdout.
- The `results['t']` attribute of the collector's results dictionary should contain a `captured_stdout` key with a value of `None`.
- The `collector.results` dictionary should still contain the original `outcome`, `when`, `passed`, and `failed` attributes.
- The `report` object should not have any additional attributes after calling `collector.handle_runtest_logreport(report)`.
- The `wasxfail` attribute of the report object should be deleted (if present).
- The `collector.results` dictionary should still contain all other attributes as before.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

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

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(result, report)` function call passes a `report` object with `capstderr=True`.

COVERAGE

src/pytest_llm_report/collector.py	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264, 268-269)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `capture_output` method captures stdout correctly.

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

Key Assertions:

- The `captured_stdout` attribute of the `TestCaseResult` object should be set to 'Some output'.
- The `report.capstdout` method should have been called with the correct value ('Some output').
- The `collector._capture_output` function should have recorded the captured stdout correctly.

COVERAGE

src/pytest_llm_report/collector.py	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264-265, 268)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `test_capture_output_truncated` function truncates output exceeding max chars.

Why Needed: This test prevents a potential bug where the collector does not truncate output exceeding the specified max characters, leading to unexpected behavior or errors in downstream processing.

Key Assertions:

- The captured stdout should be truncated to 10 characters.
- The captured stderr is empty.
- The `captured_stdout` attribute of the result object contains only 5 characters (i.e., '1234567890').
- The `captured_stderr` attribute of the result object is empty.
- The collector does not truncate output exceeding the specified max chars when capturing stdout.
- The collector does not truncate output exceeding the specified max chars when capturing stderr.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test creates a result with item markers.

Why Needed: Prevents regression in case of item markers being added to the collector without proper setup.

Key Assertions:

- item.get_closest_marker('llm_opt_out') returns MagicMock.
- item.get_closest_marker('llm_context') returns a MagicMock object with 'complete' as its argument.
- item.get_closest_marker('requirement') returns a MagicMock object with ['REQ-1', 'REQ-2'] as its arguments.
- result.param_id is set to 'param1'.
- result.llm_opt_out is set to True.
- result.llm_context_override is set to 'complete'.
- result.requirements contains ['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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `collectors` module correctly handles ReprFileLocation when used in error representation.

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

Key Assertions:

- The `_extract_error` method returns the expected string 'Crash report' for `report.longrepr.__str__.return_value = 'Crash report'`.
- The `_extract_error` method does not crash when using `ReportError` with a `ReprFileLocation` object.
- The `_extract_error` method correctly handles cases where `str()` is used on a `ReprFileLocation` object.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 an error occurs.

Why Needed: Prevents a potential bug where the test fails with a non-string result.

Key Assertions:

- The `report.longrepr` attribute is set to 'Some error occurred'.
- The value of `_extract_error(report)` is equal to 'Some error occurred'.

COVERAGE

src/pytest_llm_report/collector.py	22 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test the `_extract_skip_reason` method when `longrepr` is set to `None`.

Why Needed: Prevents a potential bug where the test fails if `longrepr` is not provided in the report.

Key Assertions:

- The method does not throw an exception or raise an error when `longrepr` is `None`.
- The `_extract_skip_reason` method returns `None` as expected when `longrepr` is `None`.
- The test asserts that `collector._extract_skip_reason(report)` returns `None` instead of raising an exception.
- The test checks if the `report.longrepr` attribute is set to `None` before calling `_extract_skip_reason`.
- If `longrepr` is not `None`, the method should return a specific value (e.g., `None`) or raise an error.
- The test verifies that the returned value is consistent across different scenarios.
- The test ensures that the `_extract_skip_reason` method behaves as expected in edge cases (e.g., when `longrepr` is not provided).

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250, 252)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 representing the skip reason.

Why Needed: This test prevents a potential regression where the skip reason is not correctly extracted from the report.

Key Assertions:

- The expected value of `report.longrepr` should be 'Just skipped'.
- The `_extract_skip_reason` method should return the correct string representation of the skip reason.
- If the test data changes, this test should still pass and not regress.

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: 123, 163)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `extract_skip_reason_tuple` method correctly extracts a skip message from a tuple containing file, line, and message information.

Why Needed: This test prevents a bug where the `extract_skip_reason_tuple` method does not extract the correct reason for skipping a test.

Key Assertions:

- The `report.longrepr` attribute is a tuple containing 'file', 'line', and 'message' values.
- The `report.longrepr` attribute contains the string 'Skipped for reason'.
- The extracted reason matches the expected reason in the `report.longrepr` attribute.

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: 123, 163)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test 'handle_collection_report_failure' verifies that the TestCollector class correctly handles a collection report failure by recording an error and updating its collection errors list.

Why Needed: This test prevents a potential bug where the TestCollector class does not handle collection reports failures correctly, potentially leading to incorrect or missing error messages in the collection errors list.

Key Assertions:

- The 'collection_errors' attribute of the collector should be updated with an instance of MagicMock.
- The 'nodeid' attribute of the first element in 'collector.collection_errors' should match the value specified in the test report.
- The 'message' attribute of the first element in 'collector.collection_errors' should match the expected error message '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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test 'handle_runtest_rerun' verifies that the `rerun` attribute of a report is correctly set to 1 after a rerun.

Why Needed: This test prevents regression where a rerun attribute is not correctly updated after a runtest rerun.

Key Assertions:

- res.rerun_count should be equal to 1
- res.final_outcome should be 'failed'
- report.rerun should have been set to 1
- report.wasxfail should have been deleted

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: 123, 163)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

Scenario: Test verifies that the TestCollector handles a runtest setup failure correctly by recording an error report.

Why Needed: This test prevents regression in the TestCollector's behavior when encountering a setup failure during a runtest.

Key Assertions:

- The 'outcome' of the collector's results for the test 't::f' is set to 'error'.
- The phase of the collector's results for the test 't::f' is set to 'setup'.
- The error message in the collector's results for the test 't::f' is set to '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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test verifies that the test handle_runtest_teardown_failure function correctly records an error when teardown fails after a pass.

Why Needed: This test prevents regression by ensuring that the collector reports an error when teardown fails, which is necessary for cleanup to occur.

Key Assertions:

- The `teardown_report` should have been marked as failed.
- The `res.outcome` should be set to 'error'.
- The `res.phase` should be set to 'teardown'.
- The `res.error_message` should contain the string 'Cleanup failed'.

COVERAGE

src/pytest_llm_report/collector.py	38 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-115, 124, 127-128, 130, 132-133, 135-137, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_coverage_boosters.py

3 tests

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

AI ASSESSMENT

Scenario: Verifies that the GeminiProvider correctly handles edge cases when parsing preferred models.

Why Needed: This test prevents a potential bug where the provider returns an empty list of models for an invalid model configuration.

Key Assertions:

- The function `provider._parse_preferred_models()` should return a list containing 'm1' and 'm2'.
- The function `provider._parse_preferred_models()` should return an empty list when the model configuration is None.
- The function `provider._parse_preferred_models()` should return an empty list when the model configuration is set to 'All'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	17 lines (ranges: 134, 136-139, 141-142, 391, 393, 423-430)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the rate limiter prevents over and under token limits when tokens are recorded but not requests.

Why Needed: This test prevents a potential bug where the rate limiter allows users to record excess tokens, potentially leading to abuse or unexpected behavior.

Key Assertions:

- assert limiter.next_available_in(60) > 0
- assert limiter.next_available_in(10) == 0
- assert limiter.record_tokens(50) <= requests_per_minute * tokens_per_minute
- assert limiter.record_tokens(110) > requests_per_minute * tokens_per_minute
- assert limiter.record_tokens(120) > requests_per_minute * tokens_per_minute
- assert limiter.record_tokens(130) > requests_per_minute * tokens_per_minute

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/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: 363-364, 367, 371-373)`

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 converts them to dictionaries with the expected values.

Why Needed: This test prevents regression in coverage booster models where the coverage percentage is not accurately converted from source code coverage entries.

Key Assertions:

- The 'coverage_percent' value in the resulting dictionary should be equal to the original `coverage_percent` value.
- The 'error' value in the resulting dictionary for LlmAnnotation should match the expected error message.
- The 'duration' value in the resulting dictionary for RunMeta should match the expected duration 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`	`46 lines (ranges: 71-78, 104-107, 109, 111-113, 115, 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

📄 tests/test_coverage_map.py

7 tests

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

AI ASSESSMENT

Scenario: Verify that the `CoverageMapper` class initializes correctly with a given configuration.

Why Needed: This test prevents a potential bug where the `CoverageMapper` instance does not have access to its configured settings.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify the `get_warnings` method returns a list of warnings.

Why Needed: Prevents potential issues with incorrect or missing warning data.

Key Assertions:

- The `get_warnings()` method should return a list of warnings.
- The `get_warnings()` method should not raise an exception.
- All warnings in the list should be instances of the `Warning` class.
- No exceptions should be raised when calling `get_warnings()` on a valid configuration.
- The warning data is properly formatted and consistent across all tests.
- The test does not fail if there are no warnings for a particular configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	3 lines (ranges: 44-45, 308)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `map_coverage` method returns an empty dictionary when no coverage file is present.

Why Needed: Prevents a regression where the test fails to report coverage for files without a corresponding coverage file.

Key Assertions:

- The `mapper.map_coverage()` function should return an empty dictionary.
- The `mapper.warnings` attribute should have at least one warning.
- The `Path.exists` and `glob.glob` mock functions should be called with the correct arguments to return False and an empty list, respectively.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `CoverageMapper` extracts node IDs for all phases when `include_phase=all`.

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

Key Assertions:

- The `_extract_nodeid()` method returns the correct node ID based on the phase.
- The `_extract_nodeid()` method does not return an empty string for any phase.
- The `_extract_nodeid()` method returns a node ID that matches the expected value for all phases.
- If `include_phase=all`, the coverage map should include all phases in the extracted node IDs.
- If `include_phase=all` and there are multiple nodes with the same name, only one of them should be included in the coverage map.
- The `_extract_nodeid()` method does not return a specific phase if it is not specified (e.g., `test.py::test_foo|run`).
- If `include_phase=all` and there are multiple nodes with different names, all of them should be included in the coverage map.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	11 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	4 lines (ranges: 44-45, 216-217)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test the ability to filter out setup phase when include_phase=run.

Why Needed: This test prevents a regression where the test would incorrectly extract node IDs for tests in the setup phase.

Key Assertions:

- The function _extract_nodeid() should return None for the given input.
- The function _extract_nodeid() should not include any node IDs from the 'setup' phase.
- The mapper._extract_nodeid() method should correctly filter out node IDs from the setup phase.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	9 lines (ranges: 44-45, 216, 220, 224-225, 228-230)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the `extract_nodeid` method of the `CoverageMapper` class correctly extracts a node ID from the run phase context.

Why Needed: This test prevents a potential bug where the node ID is not extracted correctly due to incorrect configuration or mismatched module paths.

Key Assertions:

- The expected node ID matches the actual node ID.
- The `nodeid` variable holds the correct value.
- The `assert` statement passes if the node ID is 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/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_coverage_map_coverage.py

17 tests

PASSED `tests/test_coverage_map_coverage.py::TestExtractContexts::test_contexts_by_lineno_exception` 1ms 5

AI ASSESSMENT

Scenario: Test Extracting Contexts when `contexts_by_lineno` raises an exception.

Why Needed: Prevents a potential bug where the test fails due to an unhandled exception in `contexts_by_lineno`.

Key Assertions:

- The function `_extract_contexts` should not raise an exception when called with a mock data object that has already handled the exception.
- The function `_extract_contexts` should correctly handle the exception and return an empty dictionary.
- The function `_extract_contexts` should not throw any additional errors or warnings if the `contexts_by_lineno` raises 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/coverage_map.py	29 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 137-140, 144, 148, 150, 152, 156, 160-162, 167-170, 199, 202)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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_coverage.py::TestExtractContexts::test_no_measured_files` 1ms 4

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	7 lines (ranges: 44-45, 118, 121-122, 127-128)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestExtractContexts::test_skip_non_python_files` 1ms 5

AI ASSESSMENT

Scenario: Test that non-Python files are skipped.

Why Needed: Prevents a regression where non-python files are included in coverage reports.

Key Assertions:

- mocked mock_data.measured_files returns an empty list when the input is not a Python file.
- mocked mock_data.contexts_by_lineno returns an empty dictionary when the input is not a Python file.
- The test asserts that the result of _extract_contexts is an empty dictionary for non-Python files.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestLoadCoverageData::test_coverage_not_installed` 1ms 4

AI ASSESSMENT

Scenario: Test that coverage.py is installed before attempting to load coverage data.

Why Needed: This test prevents a potential regression where coverage.py is missing, causing the _load_coverage_data method to fail and return None.

Key Assertions:

- The mapper variable is not None after calling _load_coverage_data.
- The _load_coverage_data method does not raise an ImportError when coverage.py is installed.
- The _load_coverage_data method returns a Config object instead of None when coverage.py is installed.
- The mapper variable is not None after calling _load_coverage_data() and the config is successfully created.
- The mapper variable is not None after calling _load_coverage_data() and the config is successfully created, but without coverage data.
- The mapper variable is not None after calling _load_coverage_data() and the config is successfully created, with an empty coverage data 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/coverage_map.py	2 lines (ranges: 44-45)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestLoadCoverageData::test_no_coverage_file` 1ms 5

AI ASSESSMENT

Scenario: Test 'test_no_coverage_file' verifies that when no .coverage file exists, the function returns None and generates warnings.

Why Needed: This test prevents a regression where the function would return an error or raise an exception when there is no coverage data.

Key Assertions:

- The function `mapper._load_coverage_data()` should be called with no arguments.
- The result of `mapper._load_coverage_data()` should be None.
- Any warnings generated by the mapper should contain 'W001'.
- No error or exception should be raised when there is no coverage data.
- The function `Config` should have been successfully instantiated.
- The function `CoverageMapper` should have been successfully instantiated and created a mapper object.
- The function `mapper.warnings` should not contain any warnings for the test 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/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestMapSourceCoverage::test_analysis_exception_handling` 2ms 5

AI ASSESSMENT

Scenario: Test the exception handling of analysis2 when it raises an exception.

Why Needed: This test prevents regression where analysis2 raises an exception and coverage is not handled correctly.

Key Assertions:

- The function `map_source_coverage` should return an empty list when `analysis2` raises an exception.
- A warning with the message 'COVERAGE_ANALYSIS_FAILED' should be added to the warnings list.
- All warnings in the map source coverage should have the message 'COVERAGE_ANALYSIS_FAILED'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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_coverage.py::TestMapSourceCoverage::test_empty_statements` 1ms 5

AI ASSESSMENT

Scenario: Test that an empty source file is handled correctly by the CoverageMapper.

Why Needed: This test prevents a regression where coverage reports are missing for files with no statements.

Key Assertions:

- The function `map_source_coverage` should return an empty list when given a mock Covariance object and mock data.
- The `mock_data.measured_files.return_value` should be set to `['/project/src/empty.py']` before the test is run.
- The `mock_cov.get_data.return_value` should be set to the mock data object.
- The `mock_cov.analysis2.return_value` should contain an empty list of files, a list of empty strings, and a list of empty lists.
- The function `map_source_coverage` should not raise any exceptions when given a mock Covariance object and mock data.
- The test result should be an empty list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	18 lines (ranges: 44-45, 243-244, 246-248, 250, 252-254, 259-261, 273-274, 299-300)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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_coverage.py::TestMapSourceCoverage::test_include_test_files_when_not_configured` 2ms 6

AI ASSESSMENT

Scenario: Test that test files are included when omit_tests_from_coverage is False.

Why Needed: Prevents regression in case the configuration is not properly set, where missing test files would cause coverage to be incomplete or inaccurate.

Key Assertions:

- The mapper returns exactly one result with a covered percentage of 2 out of 3 files.
- The mapper returns an empty list for all files.
- The mapper returns a single entry with the correct covered and missed counts.

COVERAGE

src/pytest_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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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_coverage.py::TestMapSourceCoverage::test_skip_non_python_files` 1ms 4

AI ASSESSMENT

Scenario: Test that non-Python files are skipped.

Why Needed: Prevents regression in coverage reporting when non-python files are present.

Key Assertions:

- The `mock_data.measured_files` method is called with the correct arguments.
- The `mock_data.contexts_by_lineno` method is called with the correct arguments.
- The `result` variable is set to an empty dictionary.
- The `assert` statement checks if the `result` is equal to the expected output.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	10 lines (ranges: 44-45, 243-244, 246-249, 299-300)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestMapSourceCoverage::test_skip_test_files_when_configured` 1ms 5

AI ASSESSMENT

Scenario: Test that test files are skipped when omit_tests_from_coverage is True.

Why Needed: This test prevents a regression where the coverage map is not generated for test files even when omit_tests_from_coverage is enabled.

Key Assertions:

- The `map_source_coverage` method of `CoverageMapper` should return an empty list if `omit_tests_from_coverage` is True and there are no test files to cover.
- The `get_data` method of the mock `CoverageMapper` object should not have been called with a non-empty list of test files when `omit_tests_from_coverage` is True.
- The `mock_cov.get_data.return_value` call should return a mock data object that has no measured files.
- The `mock_data.measured_files.return_value` call should be empty, indicating no test files were covered.
- The `mock_cov.get_data.return_value` call should not have called any methods on the mock data object when `omit_tests_from_coverage` is True.
- The `map_source_coverage` method of `CoverageMapper` should return an empty list even if there are multiple test files in the repository.
- The `map_source_coverage` method of `CoverageMapper` should not have generated any coverage for test files when `omit_tests_from_coverage` 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/coverage_map.py	15 lines (ranges: 44-45, 243-244, 246-248, 250, 252-255, 257, 299-300)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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_coverage.py::TestPhaseFiltering::test_extract_nodeid_all_phase_config` 1ms 4

AI ASSESSMENT

Scenario: Test that all phases are accepted when configured.

Why Needed: Prevents regression in case of phase filtering configuration change.

Key Assertions:

- The mapper should return the same nodeid for each phase.
- The mapper should accept any phase as input and return a matching nodeid.
- The mapper should handle phases with different module names correctly (e.g., 'test_foo.py::test_bar').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestPhaseFiltering::test_extract_nodeid_empty_string` 1ms 4

AI ASSESSMENT

Scenario: The test verifies that an empty string does not return a node ID.

Why Needed: This test prevents a potential bug where an empty string could be returned as a node ID, potentially causing issues with coverage analysis.

Key Assertions:

- assert mapper._extract_nodeid('') == None
- assert mapper._extract_nodeid(None) == 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/coverage_map.py	4 lines (ranges: 44-45, 216-217)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestPhaseFiltering::test_extract_nodeid_none` 1ms 4

AI ASSESSMENT

Scenario: Tests that None input to _extract_nodeid method returns None.

Why Needed: Prevents a potential bug where the method does not handle None inputs correctly.

Key Assertions:

- The method should return None when passed None as an argument.
- None is expected to be returned for the given input.
- The method's behavior is not tested with None inputs, which could lead to unexpected results or errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	4 lines (ranges: 44-45, 216-217)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestPhaseFiltering::test_extract_nodeid_run_phase_default` 1ms 4

AI ASSESSMENT

Scenario: Test that run phase is the default filter.

Why Needed: Prevents a regression where nodeids are not extracted for phases other than 'run'.

Key Assertions:

- mapper._extract_nodeid('test_foo.py::test_bar|run') == 'test_foo.py::test_bar'
- mapper._extract_nodeid('test_foo.py::test_bar|setup') is None
- mapper._extract_nodeid('test_foo.py::test_bar|teardown') 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/coverage_map.py	12 lines (ranges: 44-45, 216, 220, 224-225, 228-231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestPhaseFiltering::test_extract_nodeid_setup_phase_config` 1ms 4

AI ASSESSMENT

Scenario: Test that setup phase is correctly filtered when configured.

Why Needed: This test prevents a bug where the setup phase is incorrectly filtered, potentially leading to false positives or negatives in coverage reports.

Key Assertions:

- mapper._extract_nodeid('test_foo.py::test_bar|setup') == 'test_foo.py::test_bar'
- mapper._extract_nodeid('test_foo.py::test_bar|run') is None
- mapper._extract_nodeid('test_foo.py::test_bar|teardown') 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/coverage_map.py	12 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231-233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestPhaseFiltering::test_extract_nodeid_teardown_phase_config` 1ms 4

AI ASSESSMENT

Scenario: Test that teardown phase is correctly filtered when configured.

Why Needed: Prevents a potential bug where the test extract_nodeid function does not filter out nodeids from the teardown phase configuration.

Key Assertions:

- The function should return the expected nodeid when the phase matches.
- The function should return None when the phase doesn't match.
- The function should ignore nodes in the run and setup 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	12 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231, 233-234, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_coverage_map_coverage.py::TestPhaseFiltering::test_extract_nodeid_without_pipe` 1ms 4

AI ASSESSMENT

Scenario: The test verifies that the `extract_nodeid` method returns the node ID without a phase delimiter when the context is in the correct format.

Why Needed: This test prevents a potential bug where the `extract_nodeid` method incorrectly handles contexts with phase delimiters.

Key Assertions:

- The node ID should be returned unchanged (i.e., `test_foo.py::test_bar`) without any modifications.
- No additional context is required to extract the node ID (e.g., no pipe or delimiter in the file path).
- The method should not modify the original file path, only return the node ID.
- Any leading or trailing whitespace in the file path should be preserved.
- The `extract_nodeid` method should not add any additional information to the node ID (e.g., module name, filename).

COVERAGE

`src/pytest_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`	`6 lines (ranges: 44-45, 216, 220, 224, 239)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 123, 163)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

📄 tests/test_coverage_map_maximal.py

9 tests

PASSED `tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_extract_contexts_full_logic` 15ms 6

AI ASSESSMENT

Scenario: Test should extract all contexts for full logic coverage.

Why Needed: This test prevents regression in case of minimal coverage on app.py files.

Key Assertions:

- The function _extract_contexts returns 'test_app.py::test_one' and 'test_app.py::test_two' as expected.
- The line count for each context is correct (2 lines for both).
- The file paths of the contexts are correctly matched with app.py files.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 correctly handles data with no test contexts.

Why Needed: Prevents regression in coverage analysis, as it ensures that all files are covered by context.

Key Assertions:

- mock_data.measured_files.return_value should be equal to `[]`
- mock_data.contexts_by_lineno.return_value should be an empty dictionary
- result should be equal to `None` or an empty dictionary

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	14 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 144-146)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test the `CoverageMapper` class with different phases and scenarios.

Why Needed: This test prevents a bug where missing lines in the code are not correctly identified as covered by the coverage report.

Key Assertions:

- The `_extract_nodeid` method should return the expected node ID for each line of code.
- The `None` assertion is used when no lines match any phase, ensuring that the test covers all scenarios.
- The context without a pipe (`test.py::test_no_phase`) should also be correctly identified as covered by the coverage 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/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test the function "test_load_coverage_data_no_files" to ensure it correctly handles cases where no coverage files exist.

Why Needed: This test prevents a potential bug that would cause the function to incorrectly report warnings for non-existent .coverage files.

Key Assertions:

- The function should return None when no .coverage files are found.
- The number of warnings should be 1.
- The first warning message should be 'W001'.
- The current working directory should be changed to the temporary directory before calling _load_coverage_data().
- The function should not throw an exception or raise an error when no .coverage files are found.
- The function should return None immediately after checking the current working directory and calling _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	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function should return None when an error occurs while reading coverage data.
- Any warnings with messages 'Failed to read coverage data' should be raised.
- The coverage file should be created in the temporary directory before attempting to load it.
- The CoverageMapper class's _load_coverage_data method should raise an Exception when a corrupt coverage file is encountered.
- The function should not attempt to load coverage data from a non-existent or corrupted file.
- The function should handle errors raised by the mock CoverageData object correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 it correctly updates the CoverageData instances.

Why Needed: This test prevents regression in handling parallel coverage files, ensuring that the CoverageMapper class can correctly update its internal state when dealing with such files.

Key Assertions:

- ...
- ...
- ...
- ...
- Mocking mock_parallel_data1 and mock_parallel_data2 to return different instances of CoverageData.
- The update method of CoverageData should be called at least twice during the load process for parallel coverage files.
- The assert statement should not raise an AssertionError when the update method is called multiple times.
- ...
- ...
- ...

COVERAGE

src/pytest_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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test verifies the behavior of `map_coverage` method when `_load_coverage_data` returns None.

Why Needed: Prevents a potential bug where the method does not handle cases with missing coverage data correctly.

Key Assertions:

- The method should return an empty dictionary when no coverage data is loaded.
- No exception should be raised when no coverage data is loaded.
- The method should check if `_load_coverage_data` returns None before attempting to map coverage.
- The method should handle the case where `None` is returned from `_load_coverage_data` correctly.
- The method should not attempt to access any attributes or methods of the coverage data that are not present.
- The method should return a dictionary with an empty list as its value when no coverage data is loaded.
- The method should not raise an exception when no coverage data is loaded.

COVERAGE

src/pytest_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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 a potential bug where an error during analysis causes the test to fail and report incorrect results.

Key Assertions:

- The `map_source_coverage` method should not return any entries when an error occurs during analysis.
- The error message from `analysis2` should be 'Analysis failed'.
- The number of returned entries should be 0, indicating no errors in the source files.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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: Tests the coverage mapper's ability to map source code coverage across different analysis methods.

Why Needed: This test prevents regression in coverage reporting when using multiple analysis methods (analysis2) with the same data.

Key Assertions:

- The function `map_source_coverage` should return a list containing one entry with file path 'app.py', three statements, two covered lines, one missed line, and an accuracy of 66.67%.
- The assertions for each entry in the returned list should match the expected coverage statistics.

COVERAGE

src/pytest_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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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)

📄 tests/test_errors.py

3 tests

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

AI ASSESSMENT

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

Why Needed: To prevent a warning about missing coverage files for unknown warnings.

Key Assertions:

- The function returns an instance of WarningCode.W001_NO_COVERAGE with the specified code and message.
- The `detail` attribute is set to 'test-detail'.
- The `message` attribute contains the expected string 'Unknown warning.'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 bug where the warning code values are not correctly identified, potentially leading to incorrect handling of warnings in the code.

Key Assertions:

- {'name': 'assert WarningCode.W001_NO_COVERAGE.value == "W001"', 'description': 'Verify that the correct value is assigned to WarningCode.W001_NO_COVERAGE.'}
- {'name': 'assert WarningCode.W101_LLM_ENABLED.value == "W101"', 'description': 'Verify that the correct value is assigned to WarningCode.W101_LLM_ENABLED.'}
- {'name': 'assert WarningCode.W201_OUTPUT_PATH_INVALID.value == "W201"', 'description': 'Verify that the correct value is assigned to WarningCode.W201_OUTPUT_PATH_INVALID.'}
- {'name': 'assert WarningCode.W301_INVALID_CONFIG.value == "W301"', 'description': 'Verify that the correct value is assigned to WarningCode.W301_INVALID_CONFIG.'}
- {'name': 'assert WarningCode.W401_AGGREGATE_DIR_MISSING.value == "W401"', 'description': 'Verify that the correct value is assigned to WarningCode.W401_AGGREGATE_DIR_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: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test Warning.to_dict() method.

Why Needed: Prevents a warning from being silently ignored when the 'detail' field is missing in a Warning object.

Key Assertions:

- The 'code' field of the Warning object should be set to 'W001'.
- The 'message' field of the Warning object should be set to 'No coverage'.
- The 'detail' field of the Warning object should be set to 'some/path'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	6 lines (ranges: 70-72, 74-76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_errors_maximal.py

6 tests

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

AI ASSESSMENT

Scenario: Test verifies that a warning is created with the correct code and message for known code.

Why Needed: This test prevents a potential regression where warnings are not correctly generated for known code.

Key Assertions:

- The function `make_warning` returns an instance of `WarningCode.W101_LLM_ENABLED` with the expected value.
- The warning message is set to `WARNING_MESSAGES[WarningCode.W101_LLM_ENABLED]` as expected.
- The detail attribute is not provided, which is incorrect for warnings related to known code.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Tests for the `make_warning` function to handle unknown code warnings.

Why Needed: Prevents a potential bug where an unknown code is used in a warning message without being allowed by the enum.

Key Assertions:

- The function `make_warning(missing_code)` should return a warning message that indicates 'Unknown warning.'
- The value of `WARNING_MESSAGES[missing_code]` should be set to the old message before restoring it.
- The test should fail when trying to make a warning with an unknown code.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test 'test_make_warning_with_detail' verifies that a warning is created with the correct code and detail.

Why Needed: This test prevents a potential regression where a warning might not be created correctly when the detail parameter is set to an invalid configuration value.

Key Assertions:

- The function `make_warning` returns a Warning object with the specified code and detail.
- The attribute `code` of the Warning object matches the expected WarningCode.W301_INVALID_CONFIG.
- The attribute `detail` of the Warning object matches the expected string 'Bad value'.
- The warning is created correctly even if the input configuration value is 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/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that all WarningCode enum values are strings and start with 'W'.

Why Needed: Prevents a potential TypeError when trying to use non-string values as WarningCode enum members.

Key Assertions:

- assert isinstance(code.value, str) checks if the value of each WarningCode enum member is indeed a string.
- assert code.value.startswith('W') checks if all WarningCode enum members start with 'W'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test the warning to dictionary conversion without detail.

Why Needed: To prevent a potential bug where warnings are not properly serialized to JSON with detailed information.

Key Assertions:

- The `to_dict()` method of the Warning class returns a dictionary with the correct keys and values.
- The 'code' key in the returned dictionary contains the warning code.
- The 'message' key in the returned dictionary contains the warning message.
- The 'detail' field is missing from the returned dictionary.
- The 'level' field is missing from the returned dictionary.
- The 'category' field is missing from the returned dictionary.
- The 'filename' field is missing from the returned dictionary.
- The 'lineno' field is missing from the returned dictionary.
- The 'module' field is missing from 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/errors.py	5 lines (ranges: 70-72, 74, 76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 detailed information.

Why Needed: This test prevents a potential bug where warnings are not properly serialized into dictionaries due to missing or incomplete detail.

Key Assertions:

- The `to_dict()` method of the `Warning` class returns a dictionary with the correct keys ('code', 'message', and 'detail').
- The value of each key in the returned dictionary matches the expected values (e.g., 'W001' for 'code', 'No coverage' for 'message', and 'Check setup' for 'detail').
- The presence of any missing or incomplete detail information is not detected by this test, as it relies on the `to_dict()` method to handle such cases.
- Without this test, warnings may be incorrectly serialized into dictionaries, potentially leading to unexpected behavior or errors in downstream applications.

COVERAGE

src/pytest_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: 363-364, 367, 371-373)

📄 tests/test_fs.py

12 tests

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 non-python files as such.

Key Assertions:

- assert is_python_file('foo/bar.txt') is False
- assert is_python_file('foo/bar.pyc') 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: 363-364, 367, 371-373)
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 an appropriate value when given a non-`.py` file path.
- The function should correctly handle cases where the file name is not exactly `.py` but still contains Python code (e.g., `foo.py`)
- The function should not incorrectly identify files with non-Python extensions (e.g., `.txt`, `.js`) as Python files
- The function should raise an error when given a path to a non-existent file (e.g., `non_existent.py`)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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: Verify that the function returns a normalized path when there is no base directory.

Why Needed: This test prevents regression in case the `make_relative` function does not correctly handle cases where the input path has no base.

Key Assertions:

- The output of `make_relative('foo/bar')` should be `'foo/bar'`.
- The function should not raise an exception when given a non-existent directory.
- The function should return an empty string for paths with only one level of nesting.
- The function should handle cases where the input path is an absolute path.
- The function should handle cases where the input path is a relative path and does not have a base.
- The function should correctly normalize the output path even if it has multiple levels of nesting.
- The function should preserve the original directory structure when making relative paths.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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/test_fs.py::TestNormalizePath::test_already_normalized

Why Needed: Prevents a potential bug where the `normalize_path` function does not correctly handle already-normalized paths.

Key Assertions:

- The input path is already normalized (i.e., no leading or trailing slashes).
- The normalized path remains unchanged (i.e., no changes are made to the original path).
- The function returns the original path as expected when given an already-normalized path.
- The function correctly handles paths with leading/trailing slashes.
- The function does not modify the input path in any way.
- The function preserves the original directory structure of the input path.
- The function raises a `ValueError` if the input path is not normalized.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function should return the original file path without any trailing slash.
- The function should not return a file path with a trailing slash if it already does not have one.
- The function should handle paths with multiple consecutive slashes correctly.
- The function should ignore leading or trailing whitespace when stripping slashes.
- The function should raise an error for invalid input (e.g., empty string, etc.).
- The function should preserve the original directory path in case of a trailing slash.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

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

AI ASSESSMENT

Scenario: The test verifies that the `should_skip_path` function correctly skips paths matching custom patterns.

Why Needed: This test prevents a potential bug where the `should_skip_path` function incorrectly includes paths in the exclusion list, causing unexpected behavior or errors.

Key Assertions:

- assert should_skip_path('tests/conftest.py', exclude_patterns=['test*']) is True
- assert should_skip_path('src/module.py', exclude_patterns=['test*']) is False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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: The 'should_skip_path' function is called with a normal path.

Why Needed: This test prevents the 'should_skip_path' function from skipping normal file system paths.

Key Assertions:

- assert should_skip_path('src/module.py') is False
- assert not os.path.isdir('src/module.py')
- assert not os.path.exists('src/module.py')
- assert not os.path.isfile('src/module.py')
- assert not os.path.isabs('src/module.py')
- assert not should_skip_path(os.path.join('/path/to/directory', '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: 363-364, 367, 371-373)
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 not 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: 363-364, 367, 371-373)
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: Test verifies whether `should_skip_path` function correctly skips `__pycache__` directories.

Why Needed: This test prevents a potential bug where the function does not skip `__pycache__` directories as intended, potentially leading to incorrect behavior or security vulnerabilities.

Key Assertions:

- The function should return `True` when given a path that is located in `foo/__pycache__/bar.pyc`.
- The function should raise an exception with a meaningful error message when given a path that is not located in `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: 363-364, 367, 371-373)
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 regression where the function `should_skip_path` incorrectly identifies venv directories.

Key Assertions:

- The 'venv' directory is not included in the list of paths to skip.
- The '.venv' directory is also not included in the list of paths to skip.
- If a path starts with 'venv', it should be skipped.
- If a path does not start with 'venv', it should not be skipped.
- This test ensures that the function correctly handles venv directories and other non-venv directories.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

📄 tests/test_fs_coverage.py

15 tests

PASSED `tests/test_fs_coverage.py::TestIsPythonFile::test_is_python_file_false` 1ms 3

AI ASSESSMENT

Scenario: Verifies that a non-.py file does not match the expected criteria.

Why Needed: Prevents a false positive assertion for non-.py files, which could lead to incorrect results or unexpected behavior.

Key Assertions:

- The function `is_python_file` should return `False` when given a non-.py file.
- The function `is_python_file` should not return `True` when given a non-.py file.
- The function `is_python_file` should handle files with extensions other than `.py` 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: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	1 lines (ranges: 79)

PASSED `tests/test_fs_coverage.py::TestIsPythonFile::test_is_python_file_true` 1ms 3

AI ASSESSMENT

Scenario: Verifies that a module file (.py) returns True.

Why Needed: Prevents False positives where non-python files are mistakenly considered Python files.

Key Assertions:

- is_python_file('module.py') == True
- is_python_file('path/to/module.py') == True
- is_python_file(Path('module.py')) == 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: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	1 lines (ranges: 79)

PASSED `tests/test_fs_coverage.py::TestMakeRelative::test_make_relative_path_not_under_base` 1ms 3

AI ASSESSMENT

Scenario: Test makes a relative path not under the base directory.

Why Needed: Prevents regression where make_relative fails when path is not relative to base.

Key Assertions:

- The function should return a normalized absolute path.
- The function should include the project name in the result.
- The function should include the file name in the result.
- path1 is not a subdirectory of path2.
- make_relative will fail when path1 is not relative to base.
- make_relative will return an absolute path for non-relative paths.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	12 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63, 65, 67)

PASSED tests/test_fs_coverage.py::TestMakeRelative::test_make_relative_success 1ms 3

AI ASSESSMENT

Scenario: Test makes relative path to file in test directory.

Why Needed: Prevents regression when making a relative path to a file within the test directory.

Key Assertions:

- The function should be able to successfully make a relative path to a file within the test directory.
- The function should not throw an error if the file does not exist in the test directory.
- The function should preserve the original file name and path.
- The function should handle cases where the parent directory of the test directory is empty.
- The function should handle cases where the test directory has been deleted or renamed.
- The function should not throw an error if the file is a subdirectory of the test directory.
- The function should preserve the original relative path when making a relative path to a file within the test 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: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64)

PASSED `tests/test_fs_coverage.py::TestMakeRelative::test_make_relative_with_none_base` 1ms 3

AI ASSESSMENT

Scenario: Verifies that the `make_relative` function correctly normalizes a path when the base is None.

Why Needed: Prevents a potential bug where the function returns an incorrect normalized path when the base is None.

Key Assertions:

- The function should return the original file name if the base is None.
- The function should not modify the original file name.
- The function should handle cases where the base directory does not exist.
- The function should raise an error for invalid input (e.g., non-string base)
- The function should preserve the relative path information (e.g., '..', '.').
- The function should return a normalized path with the correct file extension.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`src/pytest_llm_report/util/fs.py`	`7 lines (ranges: 30, 33, 36, 39, 42, 55-56)`

PASSED `tests/test_fs_coverage.py::TestNormalizePath::test_normalize_path_backslashes` 1ms 3

AI ASSESSMENT

Scenario: Verifies the normalization of a path containing backslashes.

Why Needed: Prevents potential issues in file paths where backslashes are used instead of forward slashes.

Key Assertions:

- The normalized path contains only forward slashes.
- Backslashes are converted to forward slashes.
- No leading or trailing backslashes are preserved.
- The resulting path is consistent with the original input.
- Backslash-escaped characters are handled correctly.
- Normalization of paths containing multiple consecutive backslashes is correct.
- Prevents issues in file paths where backslashes are used instead of 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: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

PASSED tests/test_fs_coverage.py::TestNormalizePath::test_normalize_path_path_object 1ms 3

AI ASSESSMENT

Scenario: Normalizing a Path object input should return the original file path.

Why Needed: This test prevents potential issues where a custom Path object is passed to the `normalize_path` function, potentially causing unexpected behavior or incorrect results.

Key Assertions:

- The `normalize_path` function should return the original file path.
- The `normalize_path` function should not modify the input Path object.
- The `normalize_path` function should raise a `TypeError` if an unsupported type is passed as a Path object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

PASSED `tests/test_fs_coverage.py::TestNormalizePath::test_normalize_path_trailing_slash` 1ms 3

AI ASSESSMENT

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

Why Needed: Prevents a potential issue where a directory is normalized to a file path, potentially causing unexpected behavior or errors in downstream applications.

Key Assertions:

- The function `normalize_path` removes any trailing slashes from input paths.
- The resulting normalized path does not contain any trailing slashes.
- The function handles both Unix-style (with a slash) and Windows-style (without a slash) directory separators correctly.
- The normalization of a path with a trailing slash is deterministic, meaning the same input will always produce the same output.
- The function raises an error if the input path is empty or contains only whitespace characters.
- The function does not modify any system paths or environment variables.
- The test covers both cases where the input path starts and ends with a slash (Unix-style) and where it starts but not ends with a slash (Windows-style).

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`src/pytest_llm_report/util/fs.py`	`5 lines (ranges: 30, 33, 36, 39, 42)`

PASSED `tests/test_fs_coverage.py::TestShouldSkipPath::test_should_not_skip_regular_path` 1ms 3

AI ASSESSMENT

Scenario: Testing that regular paths are not skipped.

Why Needed: This test prevents a potential regression where the test might skip certain regular paths due to incorrect implementation of `should_skip_path` function.

Key Assertions:

- The `should_skip_path` function should return False for regular path inputs.
- The `should_skip_path` function should not throw an exception when given a regular path input.
- The test should verify that the function correctly handles different types of file paths (regular and non-regular).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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_coverage.py::TestShouldSkipPath::test_should_skip_git 1ms 3

AI ASSESSMENT

Scenario: Verifies whether the `.git` directory is skipped by the `should_skip_path` function.

Why Needed: Prevents a regression where the test fails when running tests on a system with a `.git` directory.

Key Assertions:

- The path to the `.git/hooks/pre-commit` file is checked and it is found in the list of paths that should be skipped.
- If the `should_skip_path` function returns False for the path to the `.git/hooks/pre-commit`, an assertion error will be raised.
- The test checks if the path to the `.git/hooks/pre-commit` file exists before asserting its value.
- If the path to the `.git/hooks/pre-commit` file does not exist, an assertion error will be raised.
- The `should_skip_path` function is called with the correct path to the `.git/hooks/pre-commit` file.
- If the `should_skip_path` function returns False for the path to the `.git/hooks/pre-commit`, a message indicating that the path should be skipped is printed to the console.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

PASSED `tests/test_fs_coverage.py::TestShouldSkipPath::test_should_skip_path_starting_with_skip_dir` 1ms 3

AI ASSESSMENT

Scenario: The test verifies whether the `should_skip_path` function correctly identifies paths starting with a 'skip' directory name.

Why Needed: This test prevents regression where the function incorrectly returns False for paths that start with 'skip'.

Key Assertions:

- assert should_skip_path('venv') is True, # Test path 'venv' starts with 'skip' and should be skipped.
- assert should_skip_path('.venv') is True, # Test path '.venv' also starts with 'skip' and should be 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/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`src/pytest_llm_report/util/fs.py`	`10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)`

PASSED `tests/test_fs_coverage.py::TestShouldSkipPath::test_should_skip_pycache` 1ms 3

AI ASSESSMENT

Scenario: The test verifies whether the 'src/__pycache__/module.cpython-312.pyc' path should be skipped.

Why Needed: This test prevents a potential issue where the test suite incorrectly includes or excludes certain paths based on incorrect assumptions about their contents.

Key Assertions:

- assert should_skip_path('src/__pycache__/module.cpython-312.pyc') is True
- assert 'src' in self.test_dir
- assert '__pycache__' in self.test_dir
- assert 'module.cpython-312' in self.test_file_list
- assert 'pycache' not in self.test_file_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: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

PASSED `tests/test_fs_coverage.py::TestShouldSkipPath::test_should_skip_site_packages` 1ms 3

AI ASSESSMENT

Scenario: Verify that the 'site-packages' directory is skipped in the test.

Why Needed: This test prevents a potential regression where site-packages directories are incorrectly included in the test coverage.

Key Assertions:

- The function should return True for the given path.
- The function should check if the path is within the 'site-packages' directory.
- The function should raise an exception or perform some other error handling when encountering a site-package directory.
- The function should not include any files or directories from the site-packages directory in the test coverage.
- The function should handle paths that are not within the 'site-packages' directory correctly.
- The function should be able to skip site-packages directories with different names (e.g. '/usr/lib/python3.12/site-packages/my_package.py').
- The function should raise an error if it encounters a path that is not a valid Python package.
- The function should handle paths that are within the 'site-packages' directory 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: 363-364, 367, 371-373)`
`src/pytest_llm_report/util/fs.py`	`10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)`

PASSED `tests/test_fs_coverage.py::TestShouldSkipPath::test_should_skip_venv` 1ms 3

AI ASSESSMENT

Scenario: The test verifies whether the 'venv' directory is skipped in the given path.

Why Needed: This test prevents a potential issue where the test fails due to incorrect handling of virtual environment directories.

Key Assertions:

- assert should_skip_path('venv/lib/python3.12/site.py') is True
- assert should_skip_path('.venv/lib/python3.12/site.py') is True

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

PASSED `tests/test_fs_coverage.py::TestShouldSkipPath::test_should_skip_with_exclude_patterns` 2ms 3

AI ASSESSMENT

Scenario: Test that custom exclude patterns work to skip a path with an excluded file.

Why Needed: This test prevents a regression where the `should_skip_path` function incorrectly returns True for paths containing excluded files.

Key Assertions:

- The `should_skip_path` function should return False for the 'src/module.py' path because it contains an excluded file ('*secret*').
- The `should_skip_path` function should return True for the 'src/secret.py' path because it does not contain any excluded files.
- The `should_skip_path` function should correctly handle paths with multiple exclude patterns by returning False when no pattern matches.
- The `should_skip_path` function should raise an error if the `exclude_patterns` parameter is empty or None.
- The `should_skip_path` function should return True for a path that does not match any of the exclude patterns.
- The `should_skip_path` function should correctly handle paths with relative file names by returning False when no pattern matches.
- The `should_skip_path` function should raise an error if the `exclude_patterns` parameter is not a list or tuple.
- The `should_skip_path` function should return True for a path that contains a file name without any exclude patterns.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/fs.py	15 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116-117, 119-121, 123)

📄 tests/test_gemini_advanced.py

4 tests

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

AI ASSESSMENT

Scenario: Verify that pruning of request times clears token usage records.

Why Needed: This test prevents a regression where the rate limiter would incorrectly clear token usage records for requests made in the past.

Key Assertions:

- The length of _request_times should be greater than 0 after pruning.
- The length of _token_usage should be greater than 0 after pruning.
- _request_times should not contain any timestamps from before the last request time recorded.
- _token_usage should not contain any timestamps from before the last token usage recorded.
- All tokens should still exist in the rate limiter's cache.
- The number of requests made within a minute should be less than or equal to 10 as expected by the _GeminiRateLimitConfig.
- The total token usage should be greater than or equal to 1000 as expected by the _GeminiRateLimiter.

COVERAGE

src/pytest_llm_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: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the rate limiter prevents requests from exceeding the specified limit in a real-time manner.

Why Needed: This test prevents regression when the rate limiter is not configured to handle requests above the specified limit.

Key Assertions:

- The `next_available_in` method should return a value greater than 0.
- The `next_available_in` method should return a value less than or equal to 60.0 seconds.
- The rate limiter should not be available for at least 1 second after recording the initial request.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	26 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95, 97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 used immediately.

Why Needed: This test prevents a potential regression in the rate limiter's behavior when tokens are not used immediately, which could cause unexpected delays or errors.

Key Assertions:

- The next_available_in method should return 0 after 10 tokens have been recorded.
- The _token_usage list should contain only two elements: '20' and '10'.
- The limiter._token_usage list should not be updated until the first record is processed.

COVERAGE

src/pytest_llm_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: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the `wait_for_slot` method of `_GeminiRateLimiter` waits for a slot before sleeping.

Why Needed: This test prevents a potential race condition where multiple requests are made in quick succession, causing the limiter to sleep prematurely.

Key Assertions:

- The `wait_for_slot` method is called with the correct number of arguments (1)
- The `time.sleep` mock object is called exactly once
- The `assert` statement is executed correctly

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: 363-364, 367, 371-373)

📄 tests/test_gemini_coverage_v2.py

4 tests

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the rate limiter does not record tokens for an extended period.

Key Assertions:

- The `_token_usage` list should be empty after calling `record_tokens(0)`.
- The length of `_token_usage` should match zero.
- No new token is added to the `_token_usage` list even if no tokens are available.
- _token_usage[0] should not exceed 100 (the default limit)
- The rate limiter does not throw an exception when `record_tokens(0)` is called with a non-zero value.
- The rate limiter correctly handles cases where the input to `record_tokens` is zero or negative.
- _token_usage[1] should be equal to 100 (the default limit) after calling `record_tokens(0)`.
- No error is thrown when `record_tokens(0)` is called with a non-zero value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	6 lines (ranges: 39-42, 66-67)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the rate limiter raises an error when exceeding the daily limit.

Why Needed: This test prevents a potential bug where the rate limiter does not raise an error when exceeding the daily limit, potentially leading to unexpected behavior or errors in downstream systems.

Key Assertions:

- The function `wait_for_slot` should be called with a valid slot value (10) and raise an exception `_GeminiRateLimitExceeded` with a matching message.
- The rate limiter's internal state should not have exceeded the daily limit after calling `record_request`.
- The error raised by the rate limiter should match `_GeminiRateLimitExceeded` exactly, including the exact error message and context.
- The test should fail when running with a valid slot value (10) to simulate an exhaustion scenario.
- The rate limiter's internal state should be reset to its initial state after calling `record_request` to prevent future exhaustion scenarios.
- The function `_GeminiRateLimitConfig` should create a valid rate limit configuration object with the correct parameters.
- The function `_GeminiRateLimiter` should correctly instantiate and configure the rate limiter with the provided limits.
- The test should pass when running without any errors or exceptions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the TPM wait time fallback prevents a regression when filling up TPM with tokens.

Why Needed: This test prevents a potential regression where the rate limiter does not properly handle filling up TPM with tokens, causing unexpected behavior in subsequent requests.

Key Assertions:

- The `wait` variable is greater than 0 after calling `limiter._seconds_until_tpm_available(now, 5)`.
- Tokens used plus request tokens are both above the limit before the rate limiter attempts to wait for TPM availability.
- Token usage is not empty before attempting to wait for TPM 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/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: 363-364, 367, 371-373)

PASSED tests/test_gemini_coverage_v2.py::test_gemini_provider_rpm_cooldown 580ms 6

AI ASSESSMENT

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

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

Key Assertions:

- The 'models/gemini-pro' model should be present in the provider's cooldowns.
- The cooldown for the 'models/gemini-pro' model should be greater than 1000.0 seconds (1 minute).
- The provider should not retry the request after a first rate limit exceeded 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	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, 225-226, 235, 237-238, 242-244, 246-247, 280-283, 286, 288-296, 298-301, 303-304, 306-307, 352, 354-356, 358-359, 387-388, 391-392)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_gemini_provider.py

5 tests

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

AI ASSESSMENT

Scenario: Test that the _annotate_internal method of GeminiProvider correctly handles a rate limit retry scenario.

Why Needed: This test prevents regression in case the API rate limits are exceeded and the provider needs to retry the annotation process.

Key Assertions:

- The mock_post call count should be equal to 2 (first time with 429, second time with 200).
- The scenario of the annotation should match 'Recovered Scenario'.
- No error is raised during the annotation process. The mock_parse.return_value indicates that no error occurred.
- The provider correctly retries the annotation after a rate limit exceeded.
- The correct model list is fetched from the API even though the first call fails with 429.
- The correct number of candidates are returned in response to successful annotations.
- No exception is raised during the parsing process. The mock_parse.return_value indicates that no error occurred.

COVERAGE

src/pytest_llm_report/collector.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, 225-226, 235, 237-238, 242-244, 246-247, 280-283, 286-289, 292-296, 298-301, 303-304, 306-307, 321, 323-326, 328-331, 333-334, 336-339, 341-347, 349, 352, 354-356, 358-361, 366-369, 380-383, 387-388, 391-393, 397-398, 402-405, 407-408, 411, 414-416, 418-420, 423, 425, 427-430, 434, 436-440, 443-446, 448-449, 451-453)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that _annotate_success verifies the correct model list and usage metadata when annotating success scenarios.

Why Needed: This test prevents regression in cases where a successful annotation is reported without proper validation of the model list and usage metadata.

Key Assertions:

- The mock response from _call_gemini contains the expected 'models' array with one item.
- The mock response from _parse_response has no error.
- The scenario assertion checks that the annotation's scenario matches the expected value.
- The annotation does not report an error.
- The usage metadata is correctly reported as 100 tokens.
- The model list is correctly fetched and validated.
- The actual call to _call_gemini returns the correct JSON structure.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	173 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-223, 225-226, 235, 280-283, 286-289, 292-296, 298-301, 303-304, 306-307, 321, 323-326, 328-331, 333-334, 336, 341-349, 352, 354-356, 358-361, 366-369, 380-383, 387-388, 391-393, 397-398, 402-405, 407-408, 411, 414-416, 418-420, 423, 425, 427-430, 434, 436-440, 443-446, 448-449, 451-453)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_gemini_provider.py::TestGeminiProvider::test_availability 3ms 5

AI ASSESSMENT

Scenario: Verify the availability of the Gemini API when no token is provided.

Why Needed: Prevent a potential bug where the provider attempts to access the API without a valid token, potentially causing unexpected behavior or errors.

Key Assertions:

- The `provider._check_availability()` method returns False when no GEMINI_API_TOKEN environment variable is set.
- The `provider._check_availability()` method should return True when no GEMINI_API_TOKEN environment variable is set and the provider's configuration is correctly set to 'gemini'.
- When the `GEMINI_API_TOKEN` environment variable is not provided, the provider should attempt to use a default or fallback strategy.
- The provider should handle cases where the API token is missing or invalid when attempting to check availability.
- The provider's configuration should be able to detect and adjust for missing or invalid tokens in a way that maintains the expected behavior.
- In cases where multiple providers are used, the correct one should be selected based on the available environment variables.
- When using multiple providers with different configurations, the correct provider should be chosen based on the availability of specific environment variables.

COVERAGE

src/pytest_llm_report/collector.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, 272-273, 275)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

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

Key Assertions:

- The next_available_in method returns None when there are no available slots in the rate limiter.
- The limiter records a request before checking if it's within the allowed time frame.
- The limiter checks for available slots every time it records a new request.
- The limiter ensures that requests are not made too frequently within a short period.
- The limiter prevents exceeding the rate limit set in the configuration.
- The limiter maintains a record of all recorded requests to track usage and prevent abuse.

COVERAGE

src/pytest_llm_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: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where multiple users could exceed the allowed number of requests per minute, potentially leading to unexpected behavior or errors.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	27 lines (ranges: 39-42, 45-46, 48, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-97, 100-102)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_hashing.py

13 tests

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

AI ASSESSMENT

Scenario: Test that different configuration providers produce different hashes.

Why Needed: This test prevents a potential bug where two different configuration providers could produce the same hash, potentially leading to incorrect results or security vulnerabilities.

Key Assertions:

- The function `compute_config_hash` should return a different hash for `config1` and `config2` when their provider is different.
- The function `compute_config_hash` should not return the same hash for `config1` and `config2` when their provider is the same.
- The function `compute_config_hash` should raise an error if both `config1` and `config2` have the same provider.
- The function `compute_config_hash` should be able to handle different providers without any issues.
- The function `compute_config_hash` should not be affected by the order of the configuration providers.
- The function `compute_config_hash` should raise an error if both `config1` and `config2` have the same provider when their hash is already known.
- The function `compute_config_hash` should return a different hash for `config1` with provider 'ollama' compared to `config2` with provider 'none'.
- The function `compute_config_hash` should be able to handle different providers without any issues, including when the same provider is used for both configurations.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 96-101, 103-104)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential issue where the hash might be too long, potentially leading to incorrect comparisons or storage.

Key Assertions:

- The length of the computed hash should be exactly 16 characters.
- The hash value should not exceed 15 bytes (128 bits).
- The hash value should start with a hexadecimal string starting with '0x'.

COVERAGE

src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 96-101, 103-104)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the file hash is not consistent with the content hash due to differences in encoding or formatting.

Key Assertions:

- The computed SHA-256 hash of the file should match its content hash.
- The content hash of the file should be equal to the computed SHA-256 hash of the same data.
- The file path and file contents should have a consistent byte order.
- The file contents should not contain any bytes that are not present in the original content.
- The file contents should not contain any bytes that are present but out of order.
- The computed SHA-256 hash of the file should be equal to the computed SHA-256 hash of the same data with all zeros appended at the end.
- The computed SHA-256 hash of the file should be equal to the computed SHA-256 hash of a file with identical content but different encoding or formatting.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/hashing.py	6 lines (ranges: 32, 44-48)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the hash computation is not accurate due to incorrect file contents or formatting.

Key Assertions:

- The length of the computed hash should be exactly 64 bytes.
- The hash value should match the expected output from `compute_file_sha256()` 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: 363-364, 367, 371-373)
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

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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.

Why Needed: This test prevents a potential issue where the HMAC signature is shorter than expected due to padding or other factors.

Key Assertions:

- The length of the HMAC signature should be exactly 64 bytes.
- The HMAC signature should not be shorter than 64 bytes.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`src/pytest_llm_report/util/hashing.py`	`1 lines (ranges: 61)`

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`src/pytest_llm_report/util/hashing.py`	`1 lines (ranges: 32)`

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the hash length may be less than 64 characters, potentially causing incorrect identification of the input data.

Key Assertions:

- The length of the output hash should be exactly 64 hexadecimal characters.
- The hash value should not exceed 64 hexadecimal characters in length.
- The hash value should not be shorter than 64 hexadecimal characters in length.
- The hash value should contain all hexadecimal digits (0-9, A-F, a-f).
- No leading zeros are allowed in the output hash.
- No trailing zeros are allowed in the output hash.
- No duplicate characters are present in the output hash.
- All characters in the input data are properly encoded in hexadecimal.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 32)

PASSED `tests/test_hashing.py::TestGetDependencySnapshot::test_includes_pytest` 82ms 3

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the dependency snapshot does not include the pytest package.

Key Assertions:

- The 'pytest' package is included in the dependency snapshot.
- The 'pytest' package is present in the snapshot.
- The 'pytest' package is listed as an item in the snapshot.
- The 'pytest' package is part of the dependency information.
- The test includes pytest in its dependency information.
- Dependency snapshot includes pytest package.
- Snapshot includes pytest package as required.
- Test includes pytest in its dependency 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: 363-364, 367, 371-373)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 113-114, 116-121)

PASSED tests/test_hashing.py::TestGetDependencySnapshot::test_returns_dict 84ms 3

AI ASSESSMENT

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

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 not None and isinstance(snapshot, dict)
- snapshot is a dictionary with keys matching the expected output

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 bug where the loaded key is not correctly deserialized due to incorrect encoding of the HMAC signature in the file.

Key Assertions:

- The 'hmac.key' file contains the correct HMAC signature for the provided secret key.
- The loaded key matches the expected value from the 'hmac.key' file.
- The HMAC signature is correctly encoded in the 'hmac.key' file.
- The 'load_hmac_key' function correctly deserializes the HMAC signature from the 'hmac.key' file.
- The secret key provided to the 'Config' constructor matches the one stored in the 'hmac.key' file.
- The configuration object created with the loaded key has the correct HMAC signature.
- The HMAC signature is not present in the 'load_hmac_key' function's internal state.
- The 'load_hmac_key' function correctly handles cases where the HMAC signature is missing or corrupted.

COVERAGE

src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 verifies that the function returns None when a missing key file is provided.

Why Needed: This test prevents a potential bug where the function incorrectly returns a non-None value for a missing key file.

Key Assertions:

- The `load_hmac_key` function should raise an exception or return None when the key file does not exist.
- The `Config` class should be able to detect and handle missing key files correctly.
- The test should verify that the expected error is raised with a meaningful message.
- The test should also verify that the `load_hmac_key` function returns None for a valid config object.
- The `key` variable should be set to None after calling `load_hmac_key(config)`.
- The `assert key is None` line should fail when `key` is not None.
- The test should also verify that the `Config` class correctly raises an exception for a missing key file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/hashing.py	4 lines (ranges: 73, 76-78)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the test fails due to an incorrect assumption about the configuration.

Key Assertions:

- The `Config()` object is created without specifying any key files.
- The `load_hmac_key(config)` function returns `None` when no key file is provided.
- The `assert` statement checks for `None` as the expected value, ensuring it's not raised due to an incorrect assumption.

COVERAGE

src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/hashing.py	2 lines (ranges: 73-74)

📄 tests/test_integration_gate.py

16 tests

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

AI ASSESSMENT

Scenario: Verify that aggregation defaults are set correctly.

Why Needed: Prevents a potential bug where aggregation settings are not properly initialized.

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: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the default capture failed output is set to False when no configuration is provided.

Why Needed: This test prevents a regression where the default capture failed output was set to True without any configuration being provided.

Key Assertions:

- config.capture_failed_output is False
- assert config.capture_failed_output is False
- expected the value of config.capture_failed_output to 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: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the context mode is not set to 'minimal' when running tests, potentially causing unexpected behavior or errors.

Key Assertions:

- The function `get_default_config()` returns an instance of `TestConfigDefaults` with `llm_context_mode` set to `'minimal'`.
- The value of `llm_context_mode` in the returned configuration is equal to 'minimal'.
- The context mode is not set to 'minimal' when running tests.
- Running tests without specifying a context mode would result in unexpected behavior or errors due to this setting.
- The test configuration does not include any overrides for `llm_context_mode`.
- Specifying the correct context mode ('minimal') during testing ensures proper execution of the test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `is_llm_enabled()` method returns False for the default configuration.
- The `get_default_config()` function returns a valid configuration object.
- The `config.is_llm_enabled()` assertion checks the correct value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	4 lines (ranges: 123, 163, 252, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the `TestConfigDefaults` class correctly sets `omit_tests_from_coverage` to True when `default_omit_tests` is set to `True`.

Why Needed: This test prevents a regression where setting `default_omit_tests` to `True` does not automatically omit tests from coverage.

Key Assertions:

- The `TestConfigDefaults` class correctly sets `omit_tests_from_coverage` to True when `default_omit_tests` is set to `True`.
- The test omits all tests from coverage by default when `default_omit_tests` is `True`.
- When `default_omit_tests` is `True`, the test configuration does not include any tests in the coverage report.
- The `TestConfigDefaults` class correctly handles the case where `default_omit_tests` is set to `True` without explicitly omitting tests.
- The test covers all cases where `default_omit_tests` is `True` and ensures that tests are omitted from coverage.
- When `default_omit_tests` is `False`, the test configuration includes all tests in the coverage report.
- The `TestConfigDefaults` class correctly sets `omit_tests_from_coverage` to False when `default_omit_tests` is set to `False`.
- The test ensures that setting `default_omit_tests` to `False` does not affect the coverage 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	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `config.provider` attribute is equal to 'none'.
- The `get_default_config()` function returns a configuration with a `provider` attribute equal to 'none'.
- The `assert` statement checks if the `config.provider` value matches '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: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM 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	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the output of a full pipeline is deterministic and sorted by nodeid.

Why Needed: This test prevents regression where the output might not be deterministic or sorted correctly due to external factors like network latency or system load.

Key Assertions:

- The list of nodeids in the report should be sorted in ascending order.
- Each nodeid should appear only once in the list.
- All nodeids should be present in the list.
- Nodeids without a test result (e.g., 'z_test.py::test_z') should not be included in the output.
- The presence of duplicate nodeids should be avoided.
- No empty lists or sets should be present in the output.
- All nodeids should have a corresponding test result.
- The order of nodeids should match the sorted list provided by the tests.
- Nodeid 'z_test.py::test_z' should appear first in the list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	80 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	117 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-

	364, 470-471, 495, 497, 499-501, 503, 506)

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the test suite is empty, causing the report to be invalid.

Key Assertions:

- The total count of tests in the report should be zero.
- The summary section of the report should have a 'total' key with a value of zero.
- The data dictionary in the report should contain a 'summary' section with a 'total' key set to 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	67 lines (ranges: 235-237, 239, 241, 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516-518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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` 35ms 6

AI ASSESSMENT

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

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

Key Assertions:

- The HTML report should be created at the specified path.
- The HTML report should contain the text content 'test_pass'.
- The HTML report should include the string '
- The HTML report should contain the string 'test_pass' in its contents.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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` 60ms 7

AI ASSESSMENT

Scenario: The test verifies that the full pipeline generates a valid JSON report with the correct schema version, summary statistics, and number of tests.

Why Needed: This test prevents regression in the integration gate where the report generation is not correctly formatted or does not contain the expected data.

Key Assertions:

- The 'schema_version' key in the JSON report should be set to SCHEMA_VERSION.
- The 'summary' key in the JSON report should have a total of 3 tests, with 1 passed, 1 failed, and 1 skipped.
- The 'passed', 'failed', and 'skipped' keys in the summary should contain the correct numbers of tests.
- The 'schema_version' key should be present in the JSON report.
- The 'report_json' path should exist in the test directory.
- The 'report_html' path should also exist in the test directory.
- The data dictionary should contain the expected keys and values for the schema version, summary statistics, and number of tests.

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	80 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	133 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223,

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

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

AI ASSESSMENT

Scenario: Test that the ReportRoot has required fields.

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

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	54 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test 'test_run_meta_has_aggregation_fields' verifies that RunMeta has aggregation fields.

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

Key Assertions:

- is_aggregated is present in data
- run_count is present 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/models.py	29 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 missing certain status fields, potentially causing incorrect interpretation of its metadata.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the schema version is not correctly defined or does not match a valid semver-like format.

Key Assertions:

- SCHEMA_VERSION is defined and has a value.
- SCHEMA_VERSION contains at least one dot (.) character.
- SCHEMA_VERSION matches a semver-like format, such as '1.2.3' or 'x.x.x'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `TestCaseResult` object is missing required fields, causing inconsistent results.

Key Assertions:

- nodeid
- outcome
- duration

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`19 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

📄 tests/test_litellm_retry_coverage.py

4 tests

PASSED `tests/test_litellm_retry_coverage.py::TestLiteLLMTokenRefreshRetry::test_all_retries_exhausted` 2.00s 5

AI ASSESSMENT

Scenario: Test that the LiteLLMProvider class handles all retries exhausting and returns an annotation with an error when API calls are mocked to fail.

Why Needed: This test prevents a potential regression where the LLMTokenRefreshRetry test case fails due to exhausted retries, causing the test to timeout or produce incorrect results.

Key Assertions:

- The `LiteLLMProvider` instance should be able to mock API calls to always fail with an exception.
- The `annotate` method of the provider should return a result object with an error attribute set to the mocked exception.
- The `error` attribute of the result object should not be None when the API call is mocked to fail.
- The test source function 'test_foo()' should not be executed when the annotation returns an error.
- The context files dictionary should remain empty when the annotation returns an error.
- The mock completion function raised an exception with a message 'API error' when called in the `mock_completion` patch.
- The `LiteLLMProvider` instance's `__call__` method should not be able to complete successfully when all retries are exhausted.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175)
src/pytest_llm_report/llm/litellm_provider.py	37 lines (ranges: 37-38, 41, 60, 62, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 108, 110-111, 114, 129, 133, 135-136, 138-139, 164-168, 170-171, 175, 179-180, 183)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_litellm_retry_coverage.py::TestLiteLLMTokenRefreshRetry::test_non_401_error_no_force_refresh` 1ms 5

AI ASSESSMENT

Scenario: Test that non-401 errors don't force token refresh.

Why Needed: Prevents regression in case of non-401 error, where token refresh should not be forced.

Key Assertions:

- The `liteellm_provider` instance does not call the completion function with a status code other than 401.
- The `liteellm_provider` instance raises an exception instead of calling the completion function with a status code other than 401.
- The `liteellm_provider` instance returns None after mocking the API call to fail with 500 (not 401).
- The `liteellm_provider` instance does not raise any assertion errors when encountering a non-401 error.
- The `liteellm_provider` instance correctly handles the case where the completion function raises an exception instead of calling the completion function.
- The `liteellm_provider` instance correctly returns None after mocking the API call to fail with 500 (not 401).
- The `liteellm_provider` instance does not raise any assertion errors when encountering a non-401 error.
- The `liteellm_provider` instance correctly handles the case where the completion function raises an exception instead of calling the completion function.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175)
src/pytest_llm_report/llm/litellm_provider.py	36 lines (ranges: 37-38, 41, 60, 62, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 108, 110-111, 114, 129, 133, 135, 138-139, 164-168, 170-171, 175, 179-180, 183)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_litellm_retry_coverage.py::TestLiteLLMTokenRefreshRetry::test_retry_succeeds_after_transient_error` 6.00s 6

AI ASSESSMENT

Scenario: Test that retry succeeds after transient error.

Why Needed: To ensure that the LLM token refresh retry mechanism works correctly in the presence of transient errors.

Key Assertions:

- The test verifies that the retry process successfully completes after a transient error is encountered.
- The test ensures that the scenario is correctly set to 'test scenario' when the error occurs.
- The test verifies that the error message contains the expected content.
- The test checks for None values in the result object's error attribute.
- The test asserts that the scenario is not None.
- The test verifies that the context files are empty.
- The test checks if the mock completion function has been called exactly twice with a transient 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/litellm_provider.py	39 lines (ranges: 37-38, 41, 60, 62, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 108, 110-111, 114, 129, 133, 135-136, 164-168, 170-171, 175, 179-180, 183, 185-186, 188, 197)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_litellm_retry_coverage.py::TestLiteLLMTokenRefreshRetry::test_token_refresh_on_401` 6.12s 7

AI ASSESSMENT

Scenario: Test that 401 error triggers token refresh (lines 123-126).

Why Needed: This test prevents regression where the token refresh fails on a 401 error.

Key Assertions:

- The function `provider.annotate` should not be able to return None without calling the `test` method.
- The function `provider.annotate` should call the `test` method after token refresh.
- The function `provider.annotate` should not fail with an exception on a 401 error.
- The function `provider.annotate` should have at least two calls before returning None or failing with an exception.
- The function `provider.annotate` should return a valid response for the test source.
- The function `provider.annotate` should call the correct context files.
- The function `provider.annotate` should not fail due to an unhandled exception on the 401 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/litellm_provider.py	46 lines (ranges: 37-38, 41-42, 44-48, 60-61, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 108, 110-111, 114, 129, 133, 135-136, 164-168, 170-171, 175, 179-181, 183, 185-186, 188, 197)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/llm/token_refresh.py	28 lines (ranges: 59-60, 63-66, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132, 153-154, 156)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_llm.py

9 tests

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the provider is not correctly identified as 'GeminiProvider' even though it matches the expected model.

Key Assertions:

- The `get_provider` function returns an instance of `GeminiProvider` when the 'provider' parameter is set to 'gemini'.
- The `__class__.__name__` attribute of the returned provider instance is equal to 'GeminiProvider'.
- The provider instance has a valid model attribute that matches the expected model.
- The provider instance does not raise an exception when called with the correct configuration.
- The provider instance can be used as intended in the test code without any issues.
- The `get_provider` function is correctly implemented to handle different providers.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `get_provider` function should return an instance of LiteLLMProvider.
- The `provider` parameter should be set to 'litellm' when calling `get_provider`.
- The returned provider instance should have a class name of 'LiteLLMProvider'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	9 lines (ranges: 52-53, 245, 247, 249, 252, 257-258, 260)
src/pytest_llm_report/llm/litellm_provider.py	3 lines (ranges: 37-38, 41)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where setting the `provider` to 'none' would cause an error.

Key Assertions:

- The `get_provider` function should return an instance of `NoopProvider`.
- The `get_provider` function should not raise any exceptions when the `provider` is set to 'none'.
- The `get_provider` function should correctly handle the case where the `provider` is 'none' without throwing an error.
- The `get_provider` function should return a new instance of `NoopProvider` each time it is called with the same configuration.
- The `get_provider` function should not raise any exceptions when the `provider` is set to 'none'.
- The `get_provider` function should correctly handle the case where the `provider` is 'none' without raising an exception.
- The `get_provider` function should return a new instance of `NoopProvider` each time it is called with the same configuration.
- The `get_provider` function should not raise any exceptions when the `provider` is set to 'none'.
- The `get_provider` function should correctly handle the case where the `provider` is 'none' without raising an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that OllamaProvider is returned when the 'provider' parameter is set to 'ollama'.

Why Needed: This test prevents a potential bug where the correct provider type (OllamaProvider) is not detected.

Key Assertions:

- assert provider.__class__.__name__ == 'OllamaProvider'
- assert isinstance(provider, OllamaProvider)
- assert provider.model == 'llama3.2'
- assert provider.config.provider == 'ollama'
- assert provider.config.model == 'llama3.2'
- assert provider.config.provider == 'ollama'
- assert provider.__class__.__name__ in ['OllamaProvider', 'OllamaModelProvider']
- assert isinstance(provider, 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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test case: Unknown provider raises ValueError when getting a provider.

Why Needed: This test prevents the unknown provider from being used without proper configuration.

Key Assertions:

- The function `get_provider` should raise a `ValueError` with message 'unknown' when called with an unknown provider.
- The error message of the ValueError should contain the string 'unknown'.
- The function `get_provider` should not be able to handle unknown providers without raising an exception.
- The test should fail if the unknown provider is successfully retrieved from the provider registry.
- The function `get_provider` should raise a `ValueError` with message 'unknown' when called with an invalid or unsupported provider type.
- The error message of the ValueError should contain the string 'unknown'.
- The function `get_provider` should not be able to handle unknown providers without raising an exception.
- The test should fail if the unknown provider is successfully retrieved from the provider registry.

COVERAGE

src/pytest_llm_report/collector.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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Testing that NoopProvider implements the LlmProvider interface.

Why Needed: Prevents regression in case of a new method addition to LlmProvider without proper update to NoopProvider.

Key Assertions:

- provider should have annotate() method
- provider should have is_available() method
- provider should have get_model_name() method
- provider should have config attribute

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 when it's supposed to.

Key Assertions:

- The annotation should be of type LlmAnnotation.
- The scenario attribute of the annotation should be an empty string.
- The why_needed attribute of the annotation should be an empty string.
- The key_assertions list 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/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the `get_model_name` method of the `NoopProvider` class returns an empty string when given an empty configuration.

Why Needed: This test prevents a potential bug where the `get_model_name` method does not handle cases with empty configurations correctly, potentially leading to unexpected behavior or errors in downstream code.

Key Assertions:

- assert provider.get_model_name() == ""
- assert provider.config is None
- assert provider.model_name is None
- assert provider.model_type is None
- assert provider.name is None
- assert provider._config is None
- assert provider._model is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 66)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 issue where the provider might not be available, potentially leading to unexpected behavior or errors.

Key Assertions:

- The `is_available()` method of the `NoopProvider` instance should return True.
- The `is_available()` method of the `NoopProvider` instance should always be called before using it.
- The `is_available()` method should not raise any exceptions when called.
- The `is_available()` method should not block the execution of other methods on the same instance.
- The `is_available()` method should return True for all valid configurations.
- The `is_available()` method should return False for invalid or missing configuration 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/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_llm_annotator.py

6 tests

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

AI ASSESSMENT

Scenario: Verify that the annotation summary is printed when annotations run.

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

Key Assertions:

- The function `get_provider` from `pytest_llm_report.llm.annotator` returns a `FakeProvider` instance.
- The `annotate_tests` function prints 'Annotated X test(s) via litellm' to stdout.
- The captured output contains the expected string 'Annotated 1 test(s) via litellm'.

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

Scenario: Test that the progress report is generated correctly when annotating tests.

Why Needed: This test prevents regression where the progress report is not generated for all test cases.

Key Assertions:

- The test case should start with a message indicating the number of tests being annotated.
- The test case should include the name of the test that was annotated.
- The progress messages should be appended to the list in the correct order.
- The progress messages should not contain any extra information (e.g. provider names).
- The progress messages should only contain relevant information about the annotation process (e.g. test ID, annotation status).

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

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

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

Key Assertions:

- The 'tests/test_a.py::test_a' node should be called when running LLM annotations with opt-out disabled.
- The first test case should have an annotation without LLM optimization.
- The second test case should not have an annotation.
- The third test case should not have an annotation.
- The number of tests called should not exceed the maximum allowed by the configuration.
- The 'tests/test_b.py::test_b' node should be called when running LLM annotations with opt-out disabled and a limit set to 1.
- The second test case should not have an annotation due to the limit being 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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

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

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

Key Assertions:

- provider.calls should contain all the node IDs of the tests passed by the annotator.
- sleep_calls should be equal to [2.0] for each test, indicating that the annotator slept for exactly 2 minutes between calls to sleep.
- the time.sleep function was called only once with a value of 2.0
- provider.calls contains all node IDs of tests passed by the annotator.
- sleep_calls is equal to [2.0] for each test, indicating that the annotator slept for exactly 2 minutes between calls to sleep.
- the time.sleep function was called only once with a value of 2.0
- provider.calls contains all node IDs of tests passed by the annotator.
- the time.sleep function was called only once with a value of 2.0

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

Scenario: Test that annotation with unavailable providers skips the test.

Why Needed: To prevent regression when an unavailable provider is used for testing.

Key Assertions:

- The `is_available` method of the `UnavailableProvider` class returns False.
- The `get_provider` function from `pytest_llm_report.llm.annotator` sets the `provider` attribute to the `UnavailableProvider` instance.
- The `annotate_tests` function is called with an empty list of tests and the `UnavailableProvider` instance.
- The `is not available` message is captured in the `captured.out` variable.

COVERAGE

src/pytest_llm_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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

📄 tests/test_llm_contract.py

13 tests

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 both 'scenario' and 'why_needed' fields.

Why Needed: This test prevents regression by ensuring that the schema requires these two critical fields, which are essential for validating the functionality of the `TestAnnotationSchema`.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

Scenario: Test that AnnotationSchema.from_dict() correctly parses a dictionary with required keys.

Why Needed: Prevents data tampering and ensures consistent schema structure.

Key Assertions:

- checks password
- checks username

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/schemas.py`	`5 lines (ranges: 77-81)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/schemas.py`	`5 lines (ranges: 77-81)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

Scenario: Test that the AnnotationSchema correctly handles partial input without any additional validation.

Why Needed: This test prevents a potential regression where the AnnotationSchema does not validate for partial inputs.

Key Assertions:

- The schema is correctly initialized with the provided scenario.
- The expected why_needed value is empty, indicating no additional validation required.
- The assertion checks that the schema's scenario attribute matches the provided input.
- The assertion checks that the schema's why_needed attribute remains empty after initialization.
- Additional validation for partial inputs would require a separate annotation or validation step.
- This test ensures the AnnotationSchema is correctly handling partial inputs without any additional 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/llm/schemas.py`	`5 lines (ranges: 77-81)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

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

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

Key Assertions:

- assert 'scenario' in ANNOTATION_JSON_SCHEMA['properties']
- assert 'why_needed' in ANNOTATION_JSON_SCHEMA['properties']
- assert 'key_assertions' in ANNOTATION_JSON_SCHEMA['properties']
- assert isinstance(ANNOTATION_JSON_SCHEMA, dict)
- assert len(ANNOTATION_JSON_SCHEMA) > 0
- assert all(key in ANNOTATION_JSON_SCHEMA for key in ['scenario', 'why_needed', '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	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `AnnotationSchema` class correctly serializes to a dictionary.

Why Needed: This test prevents regression by ensuring that the `AnnotationSchema` class handles scenarios and key assertions correctly.

Key Assertions:

- assertion 1
- assertion 2
- assertion 3
- assertion 4
- assertion 5
- assertion 6
- assertion 7
- assertion 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/schemas.py`	`7 lines (ranges: 90-92, 94-96, 98)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

Scenario: The factory function `get_provider` should return a NoopProvider instance when the provider is set to 'none'.

Why Needed: This test prevents a potential regression where the factory function returns an incorrect provider type for the 'none' provider.

Key Assertions:

- The returned provider should be of type `NoopProvider`.
- The provider instance should have no attributes or methods.
- The provider instance should not have any dependencies.
- The provider instance should not call any external functions.
- The provider instance should not have any state.
- The provider instance should not perform any actions.
- The provider instance should be a singleton.
- The factory function `get_provider` should return the correct NoopProvider instance for the 'none' provider.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `NoopProvider` class might be incorrectly identified as an `LLMProvider` due to its lack of actual LLM functionality.

Key Assertions:

- The `provider` variable is assigned an instance of `LlmProvider` using `assert isinstance(provider, LlmProvider)`.
- The `provider` variable is assigned a new instance of `NoopProvider` created with the same configuration as `config` using `provider = NoopProvider(config)`.
- The `provider` variable's type is checked to be an instance of `LlmProvider` using `isinstance(provider, LlmProvider)`.

COVERAGE

`src/pytest_llm_report/collector.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: 123, 163)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

Scenario: The NoopProvider should return an empty annotation when no node is found in the contract.

Why Needed: This test prevents a regression where the NoopProvider returns an incorrect or incomplete annotation when no node is present in the contract.

Key Assertions:

- The annotation returned by the NoopProvider is empty.
- The annotation returned by the NoopProvider does not contain any relevant information about the contract.
- The annotation returned by the NoopProvider does not match the expected output of the contract's 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	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the annotate method of the provider returns an LlmAnnotation-like object with the specified scenario, why needed, and key assertions.

Why Needed: This test prevents a potential regression where the annotation result is missing certain critical checks.

Key Assertions:

- The 'scenario' attribute of the annotated result should be set to the provided scenario.
- The 'why_needed' attribute of the annotated result should contain information about what bug or regression this test prevents.
- The 'key_assertions' attribute of the annotated result should include critical checks performed by 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/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the provider handles an empty code by returning a valid result.

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

Key Assertions:

- The annotate method should not be called with an empty string as its first argument.
- The annotate method should call the provider's `annotate` method with a valid configuration and a non-empty test result.
- The annotate method should return a valid TestCaseResult object.
- The annotate method should not raise any exceptions when given an empty code.
- The annotate method should handle cases where the test has no outcome specified.
- The annotate method should update the provider's internal state correctly after calling it.
- The annotate method should not modify the test result in a way that would affect subsequent 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/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_llm_contract.py::TestProviderContract::test_provider_has_annotate_method` 1ms 7

AI ASSESSMENT

Scenario: Test that all providers have an annotate method.

Why Needed: Prevents a bug where some providers do not have the annotate method.

Key Assertions:

- The provider has an attribute named 'annotate'.
- The provider's annotate method is callable.
- All providers should have this annotation.
- If a provider does not have this annotation, it should be considered as a bug.
- The provider should raise an exception if the annotate method is missing.
- If a provider has the annotate method but its implementation is incorrect, it should still be considered as a bug.

COVERAGE

src/pytest_llm_report/collector.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/litellm_provider.py	3 lines (ranges: 37-38, 41)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_llm_providers.py

35 tests

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

AI ASSESSMENT

Scenario: The test verifies that the annotate function handles large contexts correctly.

Why Needed: This test prevents a potential regression where annotate might not work properly with very large contexts.

Key Assertions:

- The annotate function should be able to handle context sizes up to a certain threshold without throwing an error or returning unexpected results.
- The annotate function should be able to handle context sizes above the threshold by wrapping them in a default annotation.
- The annotate function should not throw an exception when given very large contexts, but instead return a default annotation.
- The annotate function should return a default annotation for context sizes greater than the threshold, rather than throwing an error or returning unexpected results.
- The annotate function should be able to handle context sizes that are multiples of the threshold without wrapping them in a default annotation.
- The annotate function should not return any errors when given very large contexts, but instead provide a meaningful default annotation.
- The annotate function should be able to handle very large contexts by using a caching mechanism or other optimization techniques.
- The annotate function should be able to handle very large contexts without significant performance impact on subsequent 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/llm/base.py	8 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175)
src/pytest_llm_report/llm/gemini.py	153 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, 237, 249-250, 280-283, 286-289, 292-296, 298-301, 303-304, 306-307, 321, 323-326, 328-331, 333-334, 352, 354-356, 358-361, 366-369, 380-383, 387-388, 391-393, 397-398, 402-405, 407-408, 411, 414-416, 418-

	420, 423-424, 434, 436-440, 443-446, 448-449, 451-453)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The LiteLLMProvider should report a missing dependency when the 'litellm' package is required but not installed.

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 will contain an error message indicating that the 'litellm' package is missing and how to install it.
- the annotation will report the correct path to install 'litellm'.
- the annotation will not report a non-existent dependency, ensuring accurate error messages.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `GeminiProvider` requires an API token and provides a meaningful error message when it's missing.

Why Needed: The current implementation does not prevent a potential bug where the `GeminiProvider` might be used with a non-existent API token, resulting in an unhandled exception or incorrect behavior.

Key Assertions:

- The `error` attribute of the annotation is set to 'GEMINI_API_TOKEN is not set'.
- The `annotation.error` attribute contains the expected error message.
- The `provider.annotate()` method correctly raises a `ValueError` when an API token is missing.
- The `test_case` function in the test case passes without raising any exceptions.
- The `CaseResult` object passed to the `annotate()` method indicates that the test passed 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/gemini.py	12 lines (ranges: 134, 136-139, 141-142, 160-161, 167-169)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

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

Key Assertions:

- The 'json' key in captured contains the expected response data.
- The 'totalTokenCount' value in the 'usageMetadata' dictionary matches the expected count.
- At least one entry exists in the '_token_usage' list of the limiter.
- The first item in the '_token_usage' list has a 'value' equal to 123.
- The 'json' key in the captured dictionary does not contain any errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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, 225-226, 235, 280-283, 286-289, 292-296, 298-301, 303-304, 306-307, 321, 323-326, 328-331, 333-334, 336, 341-349, 352, 354-356, 358-361, 366-372, 374, 376-377, 380-383, 387-388, 391-393, 397-398, 402-405, 407-408, 411, 414-416, 418-420, 423, 425, 427-430, 434, 436-440, 443-446, 448-449, 451-453)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)

src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the LLM provider annotates retries on rate limits.

Why Needed: This test prevents a potential regression where the LLM provider does not retry after hitting a rate limit.

Key Assertions:

- The method `annotate_retries_on_rate_limit` should be called with an additional argument to specify the rate limit.
- The method `annotate_retries_on_rate_limit` should return a new annotation instance with the updated rate limit.
- The method `annotate_retries_on_rate_limit` should update the LLM provider's retry count.
- The method `annotate_retries_on_rate_limit` should not raise an exception when rate limit is exceeded.
- The method `annotate_retries_on_rate_limit` should return a new annotation instance with the updated rate limit after retrying.
- The method `annotate_retries_on_rate_limit` should update the LLM provider's retry count after retrying.
- The method `annotate_retries_on_rate_limit` should not raise an exception when rate limit is exceeded after retrying.

COVERAGE

src/pytest_llm_report/collector.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, 225-226, 235, 237-238, 242-244, 246-247, 280-283, 286-289, 292, 298-301, 303-304, 306-307, 321, 323-326, 328-331, 333-334, 336-339, 341-347, 349, 352, 354-356, 358-361, 366-372, 374-375, 380-383, 387-388, 391-393, 397-398, 402-

	405, 407-408, 411, 414-416, 418-420, 423, 425, 427-430, 434, 436-440, 443-446, 448-449, 451-453)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the `annotate` method of the `GeminiProvider` class rotates models on the daily limit.

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

Key Assertions:

- The `annotate` method should rotate models on the daily limit by updating the `model_rotation` attribute.
- The `rotate_models_on_daily_limit` method should be called with the correct arguments (e.g., `daily_limit`) to update the model rotation.
- The `update_model_rotation` method should be called with the correct value (e.g., a new daily limit) to rotate models on the daily limit.
- The `rotate_models_on_daily_limit` method should not throw an exception if the daily limit is exceeded.
- The `annotate` method should update the model rotation attribute correctly after calling `rotate_models_on_daily_limit`.
- The `model_rotation` attribute should be updated with a new value after calling `rotate_models_on_daily_limit`.
- The test should pass without any errors or exceptions when running 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/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, 225-226, 235, 280-283, 286-289, 292-296, 298-301, 303-304, 306-307, 321, 323-326, 328-331, 333-334, 336, 341-347, 349, 352, 354-356, 358-361, 366-372, 374, 376, 378-383, 387-388, 391-393, 397-399, 402-405, 407-

	408, 411, 414-416, 418-420, 423, 425-426, 434, 436-440, 443-446, 448-449, 451-453)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that annotating a model with `GeminiProvider` skips daily limits.

Why Needed: This test prevents the model from being annotated when it exceeds daily limits, ensuring data quality and compliance.

Key Assertions:

- The `GeminiProvider` is not called with an annotation key when the daily limit is exceeded.
- The `GeminiProvider` does not raise an exception when the daily limit is exceeded.
- The model's metadata does not contain a 'skipped' annotation when it exceeds daily limits.
- The model's annotations do not exceed the daily limit when it is annotated correctly.
- The `GeminiProvider` does not log any errors or warnings when it skips annotating the model due to daily limits.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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, 225-226, 235, 258-260, 280-283, 286-289, 292-296, 298-301, 303-304, 306-307, 321, 323-326, 328-331, 333-334, 336, 341-347, 349, 352, 354-356, 358-361, 366-372, 374, 376, 378-383, 387-388, 391-393, 397-399, 402-405, 407-408, 411, 414-416, 418-420, 423, 425, 427-430, 434, 436-440, 443-446, 448-449, 451-453)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)

src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the annotate method correctly annotates a successful response from LiteLLM.

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

Key Assertions:

- status ok
- redirect

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	177 lines (ranges: 39-42, 45-46, 48-49, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-101, 103, 105, 107-109, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-202, 206-208, 210, 213-214, 217-222, 225-226, 235, 280-283, 286-289, 292, 298-301, 303-304, 306-307, 321, 323-326, 328-331, 333-334, 336, 341-347, 349, 352, 354-356, 358-361, 366-372, 374-383, 387-388, 391-393, 397-398, 402-405, 407-408, 411, 414-416, 418-420, 423, 425, 427-430, 434, 436-440, 443-446, 448-449, 451-453)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the exhausted model recovers after 24 hours.

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

Key Assertions:

- The recovered model should have the same accuracy as before exhaustion.
- The recovered model should have the same number of parameters as before exhaustion.
- The recovered model's loss function should be close to its original value after 24 hours.
- The recovered model's training time should decrease by a factor of 2.5 after 24 hours.
- The recovered model's memory usage should decrease by a factor of 1.25 after 24 hours.
- The recovered model's GPU memory usage should decrease by a factor of 0.75 after 24 hours.
- The recovered model's CPU memory usage should decrease by a factor of 2.5 after 24 hours.
- The recovered model's memory usage percentage should be close to 100% after 24 hours.

COVERAGE

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

src/pytest_llm_report/llm/gemini.py	190 lines (ranges: 39-42, 45-46, 48-50, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-188, 190-191, 193-194, 196, 200-208, 210-211, 213-214, 217-222, 225-226, 235, 258-260, 280-283, 286-289, 292-296, 298-301, 303-304, 306-307, 321, 323-326, 328-331, 333-334, 336, 341-347, 349, 352, 354-356, 358-361, 366-372, 374, 376, 378-383, 387-388, 391-393, 397-399, 402-405, 407-408, 411, 414-416, 418-420, 423, 425, 427-430, 434, 436-440, 443-446, 448-449, 451-453)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The `fetch_available_models` method of the `GeminiProvider` class returns an error when there are no available models.

Why Needed: This test prevents a regression where the `fetch_available_models` method returns an error when it should not.

Key Assertions:

- assert 'No available models found.' in str(getattr(self, 'provider', None).fetch_available_models())
- assert 'No available models found.' in str(getattr(self, 'provider', self._providers).fetch_available_models())

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	65 lines (ranges: 134, 136-139, 141-142, 286, 288-289, 292-296, 298-301, 303-304, 306-307, 352, 354-356, 358-361, 366-369, 380-383, 391, 393, 397-398, 402-408, 411, 414-416, 418-420, 423-424, 434, 436-438, 441-442)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `refresh_interval` attribute of the provider is set to the expected value.
- The `model_list` attribute of the provider is updated with the latest data after the specified interval.
- The provider's state is consistent with the expected behavior after the interval has passed.
- No exceptions are raised when refreshing the model list.
- The refresh interval is correctly set to a positive value.
- The `refresh_interval` attribute is not overridden by any other provider.
- The `model_list` attribute is updated with the latest data after the specified interval, even if it's 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	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, 225-226, 235, 280-283, 286-289, 292, 298-301, 303-304, 306-307, 321, 323-326, 328-331, 333-334, 336, 341-347, 349, 352, 354-356, 358-361, 366-372, 374-375, 380-383, 387-388, 391-393, 397-399, 402-405, 407-408, 411, 414-416, 418-420, 423, 425, 427-430, 434, 436-440, 443-446, 448-449, 451-453)

src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_llm_providers.py::TestLiteLLMProvider::test_401_retry_with_token_refresh 1ms 7

AI ASSESSMENT

Scenario: test_401_retry_with_token_refresh verifies that the LiteLLM provider retries on 401 after refreshing token.

Why Needed: This test prevents a regression where the provider fails to retry after token refresh, potentially causing unexpected behavior or errors.

Key Assertions:

- Verify that the provider calls `fake_completion` with the correct API key when it first encounters a 401 Unauthorized error.
- Verify that the provider calls `fake_run` with the correct token count and returns a successful status code after refreshing the token.
- Verify that the provider captures the refreshed token in the `captured_keys` list.
- Verify that the provider does not call `fake_completion` again when it encounters another 401 Unauthorized error.
- Verify that the provider correctly updates the `litellm_token_refresh_command` and `litellm_token_refresh_interval` attributes of the `Config` object.
- Verify that the provider creates a new `CaseResult` node with an outcome of 'passed' after passing the test case.
- Verify that the `annotation.error` attribute is set to None after the test passes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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	47 lines (ranges: 37-38, 41-42, 44-48, 60-61, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 108, 110-111, 114, 116, 118-121, 164-168, 170-171, 175, 179-181, 183, 185-186, 188, 197)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/llm/token_refresh.py	28 lines (ranges: 59-60, 63, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132, 153-154, 156, 160-162)

src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_llm_providers.py::TestLiteLLMProvider::test_annotate_handles_completion_error 1ms 5

AI ASSESSMENT

Scenario: The test verifies that the annotate method of LiteLLMProvider returns an error message when a completion error occurs.

Why Needed: This test prevents regression where the LLM provider does not surface completion errors correctly.

Key Assertions:

- The annotation contains the string 'boom' which is the expected error message for a completion error.
- The annotation has an 'error' key with the value 'boom'.
- The annotation has an 'completion' key with the value 'boom'.
- The annotation does not contain any other error messages or information that could indicate a completion error occurred.
- The annotation is present in the test case where it is expected to occur.

COVERAGE

src/pytest_llm_report/collector.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	32 lines (ranges: 37-38, 41, 60, 62, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 108, 110, 114, 129, 131, 164-168, 170-171, 175, 179-180, 183)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that LiteLLMProvider rejects invalid key_assertions payloads.

Why Needed: To prevent bugs where the provider accepts non-list key_assertions payloads, which could lead to unexpected behavior or errors.

Key Assertions:

- response_data must be a list
- response_data must contain at least one item with 'key_assertion' key
- response_data must not contain any items that are dictionaries themselves (i.e. not lists)
- response_data must not contain any items that are strings or other types of values
- response_data must be a valid JSON string

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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	32 lines (ranges: 37-38, 41, 60, 62, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 164-168, 170-171, 175, 179-180, 183, 185-186, 188, 190, 195)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_llm_providers.py::TestLiteLLMProvider::test_annotate_missing_dependency` 3ms 5

AI ASSESSMENT

Scenario: The LiteLLMProvider should report an error when a missing dependency is encountered.

Why Needed: To prevent the test from passing and potentially introducing a bug or regression by incorrectly reporting a missing dependency.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.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	8 lines (ranges: 37-38, 41, 80-84)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the annotate method of LiteLLMProvider correctly annotates a successful response with mock data.

Why Needed: This test prevents regressions by ensuring that the annotation method returns an LlmAnnotation object with correct scenario, why_needed, and key_assertions values.

Key Assertions:

- annotation is an instance of LlmAnnotation
- annotation.scenario matches 'Checks login'
- annotation.why_needed matches 'Stops regressions'
- annotation.key_assertions matches ['status ok', 'redirect']
- confidence of annotation is 0.8 or higher
- captured.model matches 'gpt-4o'
- captured.messages contains 'system' role and 'def test_login()' in content

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/litellm_provider.py	31 lines (ranges: 37-38, 41, 60, 62, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 164-168, 170-171, 175, 179-180, 183, 185-186, 188, 197)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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	32 lines (ranges: 37-38, 41, 60, 62, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 164-168, 170-171, 175-176, 179-180, 183, 185-186, 188, 197)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the LiteLLM provider passes a static API key to the completion call.

Why Needed: This test prevents regression in cases where the API key is not properly passed through to the completion function.

Key Assertions:

- Verify that the captured API key matches the expected value (`static-key-placeholder`).
- Verify that the `litellm_api_key` attribute of the `Config` object matches the expected value (`static-key-placeholder`).

COVERAGE

src/pytest_llm_report/collector.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	32 lines (ranges: 37-38, 41, 60, 62, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 164-168, 170-171, 175, 179-181, 183, 185-186, 188, 197)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the provider does not detect installed modules due to incorrect module imports.

Key Assertions:

- The `is_available()` method should return True when the 'litellm' module is available in the system's modules.
- The `is_available()` method should raise an exception if the 'litellm' module is not available in the system's modules.
- The provider should be able to detect the presence of the 'litellm' module even if it is imported from a different package.
- The provider should handle cases where the 'litellm' module is not found in the system's modules, but still report that it is available.
- The provider should raise an exception when the 'litellm' module is not found in the system's modules and cannot be imported.
- The provider should correctly handle cases where the 'litellm' module is installed as a package rather than a standalone module.

COVERAGE

src/pytest_llm_report/collector.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	6 lines (ranges: 37-38, 41, 205-206, 208)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_llm_providers.py::TestLiteLLMProvider::test_token_refresh_integration` 1ms 7

AI ASSESSMENT

Scenario: Test the LiteLLM provider's token refresh integration.

Why Needed: To prevent a bug where the provider does not refresh tokens in case of an authentication error, causing the API key to remain invalid.

Key Assertions:

- The 'api_key' field in the test_case output should be set to 'dynamic-token-789'.
- The 'litellm_token_refresh_command' field in the config should be set to 'get-token'.
- The 'litellm_token_refresh_interval' field in the config should be set to 3600 (1 hour).

COVERAGE

src/pytest_llm_report/collector.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	38 lines (ranges: 37-38, 41-42, 44-48, 60-61, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 164-168, 170-171, 175, 179-181, 183, 185-186, 188, 197)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/llm/token_refresh.py	25 lines (ranges: 59-60, 63, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132, 153-154, 156)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Tests the functionality of annotating fallback providers when context length error occurs.

Why Needed: This test prevents regression in LLM providers when encountering a context length error.

Key Assertions:

- Verify that the `annotate_fallbacks_on_context_length_error` method is called with the correct arguments.
- Check if the `context_length_error` exception is raised correctly.
- Verify that the fallback provider is annotated correctly with the provided arguments.
- Ensure that the annotation message contains the expected error context.
- Test that the fallback provider's behavior changes as expected when a context length error occurs.
- Verify that the LLM model's output remains unchanged despite the context length error.
- Check if any exceptions are raised during the execution of the annotated code.
- Ensure that the `annotate_fallbacks_on_context_length_error` method is called with the correct number of arguments.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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, 64-65, 71)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test OllamaProvider::test_annotate_handles_call_error verifies that the annotate method handles call errors and returns a meaningful error message.

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

Key Assertions:

- The annotation should return an error message indicating that the call failed after 2 retries.
- The error message should include the last error that occurred during the call.
- The error message should be more informative than a generic 'Failed' message.
- The test case should pass even if the call is successful (i.e., no errors occur).
- The annotation should not return an error when the system prompt is different from the user prompt.
- The annotation should handle retries correctly, returning an error after 2 retries and a more detailed error message for subsequent retries.
- The test case should be able to reproduce the error consistently across different environments.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175)
src/pytest_llm_report/llm/ollama.py	17 lines (ranges: 40-41, 47, 50, 52, 54-55, 57-59, 73, 76-77, 79-80, 82-83)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The Ollama provider reports missing httpx dependency when annotating a test case.

Why Needed: This test prevents the Ollama provider from reporting an error for missing httpx, which could lead to incorrect or misleading results in downstream analysis.

Key Assertions:

- assert annotation.error == 'httpx not installed. Install with: pip install httpx'
- provider.annotate(test, 'def test_case(): assert True')
- test_case()
- assert test_case().__name__ == 'test_case'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test annotating full flow of Ollama provider with mocked HTTP response.

Why Needed: Prevents authentication bugs by verifying the full annotation process.

Key Assertions:

- Check if the status code is OK (200)
- Validate the token in the response
- Verify that the scenario matches the test case
- Ensure the why_needed message is displayed correctly
- Confirm that the key assertions are performed 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	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, 71, 119, 121-128, 132-135, 137, 139-140)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: The test prevents a regression where the Ollama provider fails to make an API call when the provided model and prompt are invalid.

Key Assertions:

- The `url` attribute of the captured dictionary is set to `http://localhost:11434/api/generate`.
- The `json` attribute of the captured dictionary contains a valid response with keys `model`, `prompt`, `system`, and `stream`.
- The `timeout` attribute of the captured dictionary is set to 60 seconds.
- The `url` attribute in the captured dictionary matches the expected URL for the Ollama provider's API call.
- The `json` attribute in the captured dictionary contains a valid response with the correct model and prompt values.
- The `timeout` attribute in the captured dictionary is set to 60 seconds, which is within the expected timeout range.
- The `url` attribute in the captured dictionary does not contain any invalid characters or special sequences.

COVERAGE

src/pytest_llm_report/collector.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: 119, 121-128, 132-135, 137, 139-140)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Ollama provider uses default model when not specified.

Why Needed: Prevents bug where OllamaProvider is used with an empty model, causing the default model to be used instead.

Key Assertions:

- The 'model' key in the captured response should be equal to 'llama3.2'.
- The 'model' key in the captured response should not be empty.
- The 'response' key in the captured response should contain a JSON object with a single property named 'model'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	16 lines (ranges: 119, 121-128, 132-135, 137, 139-140)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the Ollama provider returns False when the server is unavailable.

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

Key Assertions:

- The function _check_availability() of the OllamaProvider instance does not raise an exception or return a specific value when the server is unavailable.
- The function _check_availability() of the OllamaProvider instance returns False when the server is unavailable.
- The provider's configuration is updated to point to a fake HTTPX instance that raises a ConnectionError when trying to connect to the server.
- The provider's internal state is not updated to reflect the server being unavailable after calling _check_availability()
- The provider does not raise an exception or return a specific value when the server is unavailable, allowing it to continue functioning as expected

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	6 lines (ranges: 92-93, 95-96, 98-99)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The Ollama provider returns False for non-200 status codes.

Why Needed: This test prevents a potential bug where the provider incorrectly reports availability when it should not.

Key Assertions:

- assert provider._check_availability() is False
- assert config.provider == 'ollama'
- assert FakeResponse().status_code != 200
- assert fake_get('https://example.com', **kwargs).status_code == 500
- assert not isinstance(provider._check_availability(), bool)
- assert provider._check_availability() is False in mock context
- assert config._mocked_httpx.get.return_value.status_code != 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/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 92-93, 95-97)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the provider fails to check availability when it's not available.

Key Assertions:

- The '/api/tags' URL is present in the provided URL.
- The response status code is 200 (OK).
- The provider returns True for the _check_availability 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	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 92-93, 95-97)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The Ollama provider should always return `is_local=True` when the `provider` parameter is set to 'ollama'.

Why Needed: This test prevents a potential bug where the provider returns `False` when it's supposed to be `True` due to an incorrect implementation.

Key Assertions:

- The `provider` attribute of the `OllamaProvider` instance should be set to `'ollama'`.
- The `is_local()` method of the `OllamaProvider` instance should return `True`.
- The `is_local()` method should not throw an exception or raise an error when called with a valid `provider` parameter.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	1 lines (ranges: 107)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the `OllamaProvider` class's `_parse_response` method returns an error message when it encounters invalid JSON in the response.

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

Key Assertions:

- The `_parse_response` method of `OllamaProvider` should raise an exception with the message 'Failed to parse LLM response as JSON' when given invalid JSON.
- The error message returned by the `_parse_response` method should include the string 'Failed to parse LLM response as JSON'.
- The test should fail if the provided `annotation.error` is not equal to '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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The Ollama provider rejects invalid key_assertions payloads.

Why Needed: This test prevents the provider from silently failing when receiving an invalid 'key_assertions' payload in its responses.

Key Assertions:

- the 'key_assertions' field must be a list
- the 'key_assertions' field should contain valid assertions

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents potential bugs where the provider may incorrectly or incompletely parse the JSON response, leading to incorrect or missing information.

Key Assertions:

- The JSON object is present and has the expected structure (i.e., it contains 'code' and 'data' keys).
- The 'code' value is a string that can be parsed as a code snippet (e.g., a Python function call or a JavaScript expression).
- The 'data' value is an object with the expected properties (i.e., it has 'type', 'value', and 'metadata' keys).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents potential issues where the provider incorrectly interprets or fails to extract JSON from such responses, potentially leading to incorrect output or errors.

Key Assertions:

- The provider should be able to successfully parse the JSON response from the given string.
- The JSON response should contain a single object with no nested objects or arrays.
- The JSON response should not contain any keys that are not present in the expected format (e.g., 'content' instead of 'data').
- Any nested objects or arrays within the JSON response should be properly serialized and escaped.
- The provider should correctly handle cases where the input string contains multiple plain Markdown fences.
- The provider should ignore any whitespace characters between the fences, which are not part of the expected format.
- The provider should raise an error if the input string does not contain a valid JSON response in a plain Markdown 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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 that may occur when parsing invalid or malformed JSON responses.

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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_models.py

29 tests

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

AI ASSESSMENT

Scenario: CoverageEntry should serialize correctly.

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

Key Assertions:

- The 'file_path' key in the dictionary matches the expected value.
- The 'line_ranges' key in the dictionary matches the expected value.
- The 'line_count' key in the dictionary matches the expected value.
- The 'file_path' key is present and has the correct value.
- The 'line_ranges' key is present and has the correct value.
- The 'line_count' key is present and has the correct value.
- The line ranges are correctly formatted (e.g. 1-3, 5, 10-15).
- No other assertions are necessary as only these three critical checks were 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/models.py	4 lines (ranges: 260-263)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the `to_dict` method of a `CoverageEntry` object correctly serializes it into a dictionary.

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

Key Assertions:

- The expected keys in the dictionary are present and have the correct values.
- The value of `file_path` is 'src/foo.py'.
- The value of `line_ranges` is a string containing comma-separated ranges, e.g. '1-3, 5, 10-15'.
- The value of `line_count` is 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`	`3 lines (ranges: 213-215)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

Scenario: Test CoverageEntry to_dict serialization correctness.

Why Needed: To ensure that the `CoverageEntry` class correctly serializes its internal data into a dictionary.

Key Assertions:

- The 'file_path' key in the dictionary should match the expected value.
- The 'line_ranges' key in the dictionary should match the expected value.
- The 'line_count' key in the dictionary should match the expected value.
- The 'file_path' key should be present in the dictionary and have the correct value.
- The 'line_ranges' key should be present in the dictionary and have the correct value.
- The 'line_count' key should be present in the dictionary and have the correct value.
- Each assertion should pass without any errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	4 lines (ranges: 40-43)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where an empty annotation is not properly initialized with default values.

Key Assertions:

- annotation.scenario == "" (empty string)
- annotation.why_needed == "Empty annotation has default values" (expected message)
- annotation.key_assertions == [] (no key assertions performed)
- assertion for annotation.confidence is None
- assertion for annotation.error is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 correctly serialized without any optional fields.

Key Assertions:

- Expected 'scenario' to be in the dictionary.
- Expected 'why_needed' to be in the dictionary.
- Expected 'key_assertions' to be in the dictionary.
- Expected 'confidence' not to be in the dictionary when it is None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	8 lines (ranges: 104-107, 109, 111, 113, 115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM 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/models.py	10 lines (ranges: 104-107, 109-111, 113-115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test the default report schema and structure.

Why Needed: Prevents a potential bug where the report's schema version is not correctly set to the latest version.

Key Assertions:

- The 'schema_version' key in the dictionary should be equal to SCHEMA_VERSION.
- The 'tests' key in the dictionary should be an empty list.
- The 'warnings' key in the dictionary should not exist.
- The 'collection_errors' key in the dictionary should 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	54 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 bug where the report does not contain all collection errors, potentially leading to incorrect reporting or missing error messages.

Key Assertions:

- The length of `collection_errors` is 1.
- The value of `nodeid` in the first `collection_error` is 'test_bad.py'.
- All elements in `collection_errors` are dictionaries with a valid `nodeid` 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/models.py	58 lines (ranges: 213-215, 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514-516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test verifies the presence of warnings in a report.

Why Needed: This test prevents a regression where reports without warnings are not properly reported.

Key Assertions:

- The 'report' object has exactly one warning.
- The first warning's code is 'W001'.
- A warning with the code 'W001' exists in the report.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	60 lines (ranges: 235-237, 239, 241, 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516-518, 520, 522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM 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/models.py	73 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 `ReportWarning` returns a dictionary with the 'detail' key.

Why Needed: This test prevents a warning about missing coverage detail in reports.

Key Assertions:

- The 'detail' key is present and contains the expected value '/path/to/file'.
- The 'detail' key does not contain any other values or errors.
- If the 'detail' key is missing, the test will fail with a `AssertionError`.
- If the 'detail' key contains other values than '/path/to/file', the test will fail with an incorrect message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	6 lines (ranges: 235-237, 239-241)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 detail.

Why Needed: This test prevents a warning from being reported when a ReportWarning object does not contain any detailed information.

Key Assertions:

- The 'detail' key should be absent in the dictionary representation of the ReportWarning object.
- The 'code' and 'message' keys should be present in the dictionary representation of the ReportWarning object.
- A warning without detail should have a 'detail' key set to 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	5 lines (ranges: 235-237, 239, 241)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that RunMeta has aggregation fields.

Why Needed: This test prevents regression where the aggregation policy is set to 'sum' instead of 'merge'.

Key Assertions:

- The run_id should be present in the dictionary.
- The run_group_id should be present in the dictionary.
- The is_aggregated attribute should be True.
- The aggregation_policy should be 'merge'.
- The run_count should be 3.
- The length of source_reports should be 2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	39 lines (ranges: 283-285, 287-289, 370-386, 388, 391, 393, 396, 399, 401, 403, 405-411, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 should not be present in the data.
- The 'llm_provider' key should not be present in the data.
- The 'llm_model' key should not 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: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression by ensuring that LLM traceability fields are present and correctly configured.

Key Assertions:

- The value of llm_annotations_enabled is True.
- llm_provider matches 'ollama'.
- llm_model matches 'llama3.2:1b'.
- llm_context_mode matches 'complete'.
- llm_annotations_count matches 10.
- llm_annotations_errors matches 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: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413-425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test 'Non-aggregated report should not include source_reports' verifies that a non-aggregated run does not include source reports.

Why Needed: This test prevents regression where the 'source_reports' key is present in the output of a non-aggregated run, potentially masking important information.

Key Assertions:

- The 'source_reports' key should be absent from the output of a non-aggregated run.
- The value of 'is_aggregated' for the meta object should be False.
- The presence of 'source_reports' in the dictionary should indicate that it is aggregated.
- If 'source_reports' were present, it would imply aggregation even if not explicitly stated.
- Including 'source_reports' could hide important information about the data source or processing steps.

COVERAGE

src/pytest_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: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 invalid metadata.

Key Assertions:

- The 'git_sha' field should be set to the provided value.
- The 'git_dirty' field should be set to True if the source report is not empty.
- The 'repo_version', 'repo_git_sha', and 'plugin_git_sha' fields should match the expected values.
- The 'config_hash' field should be set to a valid hash.
- The length of the 'source_reports' list should be 1 in this case.
- The 'pytest_invocation' field should contain the provided value.
- The 'pytest_config_summary' field should not be present in the dictionary.
- The 'run_id' and 'run_group_id' fields should match the expected values.
- The 'is_aggregated' field should be set to True if the run is aggregated.

COVERAGE

src/pytest_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: 283-285, 287-289, 370-386, 388-411, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: TestRunMeta::test_run_status_fields verifies that the RunMeta object includes all necessary run status fields.

Why Needed: This test prevents a potential regression where the RunMeta object is missing certain critical fields, potentially leading to incorrect or incomplete data.

Key Assertions:

- The 'exit_code' field should be equal to 1.
- The 'interrupted' field should be True.
- The 'collect_only' field should be True.
- The 'collected_count' field should be equal to the number of runs collected.
- The 'selected_count' field should be equal to the number of selected runs.
- The 'deselected_count' field should be equal to the number of deselected runs.

COVERAGE

src/pytest_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: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the schema version is correctly formatted (semver),

Why Needed: This test prevents a potential bug where the schema version is not in semver format, which could lead to unexpected behavior or errors.

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 have exactly three characters (digits or dots).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that the `ReportRoot` class includes the schema version in its report root structure.

Why Needed: This test prevents a potential regression 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` class should be set to the provided value.
- The `to_dict()` method of the `ReportRoot` class should return a dictionary with a `schema_version` key matching the provided value.
- The `schema_version` value in the returned dictionary should match the expected value (`SCHEMA_VERSION`).

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	54 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the coverage entry data is not properly formatted in the dictionary.

Key Assertions:

- The 'file_path' key should contain the expected value, 'src/foo.py'.
- The 'line_ranges' key should contain the expected string format, '1-3, 5, 10-15'.
- The 'line_count' key should contain the expected integer value, 10.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	8 lines (ranges: 71-78)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression by ensuring that the minimal annotation format includes all necessary information.

Key Assertions:

- The 'scenario' field is present in the returned dictionary.
- The 'why_needed' field is present in the returned dictionary.
- The 'key_assertions' field is present in the returned dictionary.
- The 'confidence' field is not included in the returned dictionary when it's `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: 283-285, 287, 289)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

Scenario: Test SourceReport to_dict_with_run_id verifies that the 'run_id' key is present in the output dictionary.

Why Needed: This test prevents a potential bug where the 'run_id' key is missing from the output dictionary, potentially causing issues with downstream processing or reporting.

Key Assertions:

- The 'run_id' key should be present in the output dictionary.
- The value of the 'run_id' key should match the provided run ID.
- The 'run_id' key should not be empty or null.
- The 'run_id' key should have the correct format (e.g., 'run-1')
- The 'run_id' key should be a string value
- The 'run_id' key should not be a number or integer value

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	6 lines (ranges: 283-285, 287-289)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_models.py::TestSummary::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 serialized data does not match the expected format, potentially leading to incorrect coverage summaries or other issues.

Key Assertions:

- The 'file_path' key in the dictionary is set to 'src/foo.py'.
- The 'line_ranges' key in the dictionary is set to '1-3, 5, 10-15'.
- The 'line_count' key in the dictionary is set to 10.
- The values of all keys are correct and match the expected format.
- The 'file_path' value does not contain any invalid characters or whitespace.
- The 'line_ranges' value does not contain any invalid characters or whitespace.
- The 'line_count' value matches the actual number of lines in the file.
- No unexpected keys are added to the dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	11 lines (ranges: 455-463, 465, 467)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test that a minimal result has the required fields.

Why Needed: This test prevents regression in case of incomplete or missing results.

Key Assertions:

- The 'nodeid' field is set to the expected value.
- The 'outcome' field is set to 'passed'.
- The 'duration' field is set to 0.0 (or any other default value).
- The 'phase' field is 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	19 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test case "Result with coverage" verifies that the `TestCaseResult` object includes a coverage list.

Why Needed: This test prevents regression where the coverage report is missing or incorrect.

Key Assertions:

- The `coverage` attribute of the `TestCaseResult` object should contain exactly one entry.
- The `file_path` attribute of the first `CoverageEntry` in the `coverage` list should be 'src/foo.py'.
- The number of entries in the `coverage` list should be 1.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	24 lines (ranges: 40-43, 162, 166-171, 173, 175, 177, 179, 182-184, 186, 188, 190, 192, 194, 196)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test Result with LLM Opt-Out should include flag.

Why Needed: This test prevents regression where the LLM opt-out flag is not properly set.

Key Assertions:

- The 'llm_opt_out' key in the result dictionary should be True.
- The 'llm_opt_out' value in the result dictionary should match the expected boolean value.
- The 'llm_opt_out' value should be present in the result dictionary.
- If LLM opt-out is not set, the 'llm_opt_out' key and value should be absent from the result dictionary.
- If LLM opt-out is set to False, the 'llm_opt_out' value should be False.
- If LLM opt-out is set to True, the 'llm_opt_out' value 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	20 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186-188, 190, 192, 194, 196)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test 'Result with reruns' verifies that the 'rerun_count' and 'final_outcome' fields are included in the result dictionary.

Why Needed: This test prevents regression where a test case is marked as failed but does not actually fail, resulting in missing 'rerun_count' and 'final_outcome' fields in the result dictionary.

Key Assertions:

- The 'rerun_count' field should be present in the result dictionary with value 2.
- The 'final_outcome' field should be present in the result dictionary with value 'passed'.
- Both 'rerun_count' and 'final_outcome' fields should 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	21 lines (ranges: 162, 166-171, 173, 175, 177, 179-182, 184, 186, 188, 190, 192, 194, 196)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where the 'rerun_count' and 'final_outcome' fields are included in the result of a test run without reruns, potentially hiding important information about the test execution.

Key Assertions:

- The 'result' dictionary should not contain 'rerun_count' key.
- The 'result' dictionary should not contain 'final_outcome' key.
- The 'result' dictionary does not include 'rerun_count' and 'final_outcome' keys.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`19 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

📄 tests/test_models_coverage.py

15 tests

PASSED `tests/test_models_coverage.py::TestReportRootToDict::test_to_dict_with_all_optional_fields` 1ms 3

AI ASSESSMENT

Scenario: Test to_dict includes all optional fields when set.

Why Needed: Prevents bar regression in coverage analysis.

Key Assertions:

- assert result['param_id'] == 'a-b-c',
- assert result['param_summary'] == 'a=1, b=2, c=3',
- assert result['captured_stdout'] == 'stdout content',
- assert result['captured_stderr'] == 'stderr content',
- assert result['requirements'] == ['REQ-100'],
- assert result['llm_opt_out'] is True,
- assert result['llm_context_override'] == 'complete',
- assert len(result['coverage']) == 1,
- assert result['llm_annotation']['scenario'] == 'Tests 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/models.py	81 lines (ranges: 71-78, 213-215, 235-237, 239, 241, 260-263, 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514-528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestReportRootToDict::test_to_dict_with_artifacts` 1ms 3

AI ASSESSMENT

Scenario: Test 'test_to_dict_with_artifacts' verifies that the `to_dict` method includes artifacts when set.

Why Needed: This test prevents a regression where the `to_dict` method does not include artifacts, potentially leading to incorrect reporting or analysis of report metadata.

Key Assertions:

- The length of 'artifacts' in the result dictionary is equal to 2.
- The path of the first artifact entry in the result dictionary matches 'report.html'.
- All artifact entries have a non-empty `path` key.
- Each artifact entry has a non-empty `sha256` and `size_bytes` keys.
- The `to_dict` method returns a dictionary with all required keys (artifacts, path, sha256, size_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/models.py	59 lines (ranges: 260-263, 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518-520, 522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestReportRootToDict::test_to_dict_with_collection_errors` 1ms 3

AI ASSESSMENT

Scenario: Test to_dict with collection errors verifies that it includes any collection_errors set in the ReportRoot.

Why Needed: This test prevents a potential bug where the to_dict method of ReportRoot does not include all collection_errors set in the report.

Key Assertions:

- The length of result['collection_errors'] is equal to 1.
- result['collection_errors'][0]['nodeid'] is equal to 'broken_test.py'.
- All collection_errors are included in the result dictionary.
- CollectionError objects are present in the collection_errors list.
- Each CollectionError object has a 'nodeid' attribute set correctly.
- The 'nodeid' value matches the expected node id.
- The message of each CollectionError object is correct and does not contain any syntax errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	58 lines (ranges: 213-215, 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514-516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestReportRootToDict::test_to_dict_with_custom_metadata` 1ms 3

AI ASSESSMENT

Scenario: Test to_dict includes custom_metadata when set.

Why Needed: Prevents a potential bug where the custom metadata is not included in the report even if it's present.

Key Assertions:

- {'message': 'custom_metadata is present and has the correct keys', 'expected_value': 'project'}
- {'message': 'custom_metadata is present and has the correct values', 'expected_value': 'staging'}
- {'message': 'custom_metadata is present and has the correct value for build_number', 'expected_value': 123}

COVERAGE

src/pytest_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	55 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522-524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestReportRootToDict::test_to_dict_with_hmac_signature` 1ms 3

AI ASSESSMENT

LLM 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/models.py	55 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526-528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestReportRootToDict::test_to_dict_with_sha256` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `to_dict` method of `ReportRoot` returns a dictionary with a 'sha256' key when set.

Why Needed: This test prevents a bug where the SHA-256 hash is not included in the report's dictionary.

Key Assertions:

- The 'sha256' key should be present in the dictionary.
- The value of the 'sha256' key should match the provided SHA-256 string.
- The 'sha256' key should be included in the dictionary even if it is not set.
- The test should fail when the 'sha256' key is not set.
- The test should pass when the 'sha256' key is set with a valid SHA-256 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	55 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524-526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestReportRootToDict::test_to_dict_with_source_coverage` 1ms 3

AI ASSESSMENT

Scenario: Test to_dict includes source_coverage when set.

Why Needed: Prevents regression in test coverage reporting.

Key Assertions:

- The 'source_coverage' key should be present in the result dictionary.
- The 'file_path' value of the first 'SourceCoverageEntry' should match 'src/mod.py'.
- All 'SourceCoverageEntry' entries under 'source_coverage' should have a valid 'file_path'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	63 lines (ranges: 71-78, 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520-522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestReportRootToDict::test_to_dict_with_warnings` 1ms 3

AI ASSESSMENT

Scenario: Test 'test_to_dict_with_warnings' verifies that the `to_dict` method includes warnings when set.

Why Needed: This test prevents a regression where warnings are not included in the report even if there is coverage data.

Key Assertions:

- The length of `result['warnings']` should be 1.
- The code 'W001' in `result['warnings'][0]` should match the expected value.
- The warning message 'No coverage data' should be present in the warning object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	60 lines (ranges: 235-237, 239, 241, 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516-518, 520, 522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestSummaryToDict::test_to_dict_with_coverage_total_percent` 1ms 3

AI ASSESSMENT

Scenario: Test to_dict includes coverage_total_percent when set.

Why Needed: Prevents a potential bug where the 'coverage_total_percent' is not included in the test results even though it's present in the Summary object.

Key Assertions:

- The 'coverage_total_percent' key should be present in the result dictionary.
- The value of 'coverage_total_percent' should match the provided value (85.5) when compared to the summary object.
- If the 'coverage_total_percent' is missing, the test will fail with an AssertionError.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	12 lines (ranges: 455-463, 465-467)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestSummaryToDict::test_to_dict_without_coverage_total_percent` 1ms 3

AI ASSESSMENT

Scenario: This test verifies that the `to_dict` method of `Summary` class excludes coverage_total_percent when it's None.

Why Needed: The test prevents a regression where the `to_dict` method returns incorrect results when there is no coverage total percent.

Key Assertions:

- summary.total and summary.passed are both set to 10
- result['coverage_total_percent'] is None
- result does not contain 'coverage_total_percent'
- result does not contain 'total' or 'passed'
- result has a different structure than expected
- the coverage total percent is missing from the result
- the test passes if summary.total and summary.passed are both 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`	`11 lines (ranges: 455-463, 465, 467)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

PASSED `tests/test_models_coverage.py::TestTestCaseResultToDict::test_to_dict_with_all_optional_fields` 1ms 3

AI ASSESSMENT

Scenario: Test to_dict includes all optional fields when set.

Why Needed: This test prevents regression in coverage calculation where llm_opt_out is True and llm_context_override is not provided.

Key Assertions:

- assert result['param_id'] == 'a-b-c'
- assert result['param_summary'] == 'a=1, b=2, c=3'
- assert result['captured_stdout'] == 'stdout content'
- assert result['captured_stderr'] == 'stderr content'
- assert result['requirements'] == ['REQ-100']
- assert result['llm_opt_out'] is True
- assert result['llm_context_override'] == 'complete'
- assert len(result['coverage']) == 1

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	41 lines (ranges: 40-43, 104-107, 109, 111, 113, 115, 162, 166-171, 173-179, 182-196)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestTestCaseResultToDict::test_to_dict_with_captured_stderr` 1ms 3

AI ASSESSMENT

Scenario: Test to_dict includes captured_stderr when set.

Why Needed: This test prevents a regression where the 'captured_stderr' is not included in the test result dictionary.

Key Assertions:

- The 'captured_stderr' key should be present in the test result dictionary.
- The value of the 'captured_stderr' key should match the captured error message.
- If no error is captured, the 'captured_stderr' key should not be present in the test 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	20 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192-194, 196)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestTestCaseResultToDict::test_to_dict_with_captured_stdout` 1ms 3

AI ASSESSMENT

Scenario: Test to_dict includes captured_stdout when set.

Why Needed: This test prevents a potential bug where the 'captured_stdout' key is not included in the result dictionary even though it was captured during the execution of the test.

Key Assertions:

- The 'captured_stdout' key should be present in the result dictionary.
- The value of the 'captured_stdout' key should match the captured output.
- If the test is successful, the 'captured_stdout' key should not be empty or None.
- If the test fails, the 'captured_stdout' key should contain a meaningful error message or exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	20 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190-192, 194, 196)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestTestCaseResultToDict::test_to_dict_with_param_summary` 1ms 3

AI ASSESSMENT

Scenario: Test to_dict includes param_summary when set.

Why Needed: To ensure the test_to_dict method correctly handles param_summary in the output dictionary.

Key Assertions:

- The 'param_summary' key is present in the result dictionary.
- The value of 'param_summary' matches the expected string 'x=1, y=2'.
- The 'param_summary' key is included in the test output even when it's empty.
- The 'param_summary' key is not included if no param_summary is provided.
- The 'param_summary' key has a correct format (e.g., 'key=value', etc.)
- The 'param_summary' key is present in the result dictionary for all test cases.
- The 'param_summary' key is present even when it's empty (e.g., for tests with no param_summary)
- The 'param_summary' key has a correct value (e.g., 'x=1, y=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	21 lines (ranges: 162, 166-171, 173, 175-179, 182, 184, 186, 188, 190, 192, 194, 196)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_models_coverage.py::TestTestCaseResultToDict::test_to_dict_with_requirements` 1ms 3

AI ASSESSMENT

Scenario: Test to_dict includes requirements when set.

Why Needed: This test prevents a regression where the 'requirements' key is missing from the TestCaseResult object.

Key Assertions:

- The 'requirements' key should be present in the TestCaseResult object.
- The 'requirements' key should contain the expected values (REQ-001 and REQ-002).
- If no requirements are provided, the 'requirements' key should still be present with an empty list.
- If a requirement is missing, it should raise an AssertionError.
- If a requirement has a different name than 'REQ-001' or 'REQ-002', it should raise an AssertionError.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`20 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194-196)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

📄 tests/test_options.py

17 tests

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

AI ASSESSMENT

Scenario: Test that default values are set correctly for a test configuration.

Why Needed: This test prevents regression in case the default values of the test configuration are not set correctly.

Key Assertions:

- cfg.provider == 'none' (default value)
- cfg.llm_context_mode == 'minimal' (default value)
- cfg.llm_max_tests == 0 (default value)
- cfg.llm_max_retries == 10 (default value)
- cfg.llm_context_bytes == 32000 (additional default value)
- cfg.llm_context_file_limit == 10 (additional default value)
- cfg.llm_requests_per_minute == 5 (additional default value)
- cfg.llm_timeout_seconds == 30 (additional default value)
- cfg.llm_cache_ttl_seconds == 86400 (additional default value)
- cfg.include_phase == 'run' (default value)
- cfg.aggregate_policy == 'latest' (default value)
- cfg.is_llm_enabled() is False (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/options.py	3 lines (ranges: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM 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`	`3 lines (ranges: 123, 163, 261)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

Scenario: Test that the `is_llm_enabled` check is enabled for providers other than 'none' and returns True for 'ollama', but disabled otherwise.

Why Needed: This test prevents a regression where the `is_llm_enabled` check was not properly configured for different provider values.

Key Assertions:

- The function `Config.is_llm_enabled()` should return False when the provider is set to 'none'.
- The function `Config.is_llm_enabled()` should return True when the provider is set to 'ollama'.
- The function `Config.is_llm_enabled()` should not return a boolean value when the provider is set to an unknown or default value (e.g. 'default').

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`3 lines (ranges: 123, 163, 252)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

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

AI ASSESSMENT

LLM 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	23 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-213, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM 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	23 lines (ranges: 123, 163, 191, 194-195, 201-205, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM 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	23 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-221, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM 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	22 lines (ranges: 123, 163, 191, 194-197, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test validation of numeric constraints for Config object.

Why Needed: Prevents regression where a Config object with invalid numeric ranges is created.

Key Assertions:

- cfg.validate() returns an empty list if all numeric constraints are met.
- cfg.validate() raises an AssertionError if any numeric constraint is not met.
- The 'llm_context_bytes' value should be at least 1000.
- The 'llm_max_tests' value should be positive or zero.
- 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 positive or zero.
- A Config object with invalid numeric ranges is created when llm_context_bytes < 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/options.py	25 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237-246, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Testing the `validate()` method of the `Config` class with a valid configuration.

Why Needed: This test prevents potential bugs or regressions where an invalid configuration is passed to the validation process.

Key Assertions:

- The `validate()` method should return an empty list (`[]`) when given a valid configuration.
- A configuration object should be created with no errors.
- No exceptions should be raised during the validation process.
- The configuration should be in a valid format (i.e., not empty or invalid).
- The `validate()` method should return an empty list (`[]`) when given a well-formed configuration.
- A `Config` object with no errors should be created and validated successfully.
- No error messages should be printed during the validation process.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test loads aggregation options for pytest configuration.

Why Needed: Prevents a bug where the aggregate policy is not correctly applied to the run ID and group ID.

Key Assertions:

- The `aggregate_dir` option should be set to 'aggr_dir'.
- The `aggregate_policy` option should be set to 'merge'.
- The `aggregate_run_id` option should match 'run-123'.
- The `aggregate_group_id` option should match '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	29 lines (ranges: 123, 163, 276, 279-280, 288-290, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440-448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test handling when pyproject.toml doesn't exist.

Why Needed: This test prevents a bug where the LLM report generation fails due to missing pyproject.toml file.

Key Assertions:

- The `llm_max_retries` attribute in the configuration should be set to 10 by default.
- The `llm_report_html`, `llm_report_json`, and `llm_report_pdf` attributes should not have any values assigned when pyproject.toml is missing.
- The `llm_evidence_bundle` attribute should remain unchanged with a value of 'None'.
- The `llm_dependency_snapshot` attribute should be set to an empty list.
- The `llm_requests_per_minute` attribute should not have any values assigned when pyproject.toml is missing.
- The `llm_aggregate_dir`, `llm_aggregate_policy`, and `llm_aggregate_run_id` attributes should remain unchanged with default values.
- The `llm_coverage_source` attribute should be set to an empty string.
- The `llm_model` attribute should not have any values assigned when pyproject.toml is missing.
- The `llm_context_mode` attribute should be set to 'test' by default.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	25 lines (ranges: 123, 163, 276, 279-280, 288-290, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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' when loading coverage.

Why Needed: This test prevents a potential bug where the `llm_coverage_source` option is not correctly set, potentially leading to incorrect coverage analysis.

Key Assertions:

- The value of `cfg.llm_coverage_source` should be 'cov_dir'.
- The `llm_coverage_source` option should have been successfully updated with the specified value.
- The test should fail if the `llm_coverage_source` option is not set to 'cov_dir' after updating it.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	26 lines (ranges: 123, 163, 276, 279-280, 288-290, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448-449, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the user is not able to load configuration with default settings.

Key Assertions:

- The 'provider' attribute of the configuration object should be set to 'none'.
- The 'report_html' attribute of the configuration object should be None.
- The provider and report HTML values should match the expected default 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	25 lines (ranges: 123, 163, 276, 279-280, 288-290, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options.py::TestLoadConfig::test_load_from_cli_overrides_pyproject` 2ms 3

AI ASSESSMENT

Scenario: Testing the test_load_from_cli_overrides_pyproject function to verify that CLI options override pyproject.toml options.

Why Needed: This test prevents a potential regression where CLI options are not overriding pyproject.toml options, potentially leading to unexpected behavior or errors in the application.

Key Assertions:

- The '--pyproject' option overrides the 'pyproject.toml' configuration file.
- The '--pyproject-repo' option overrides the 'pyproject.toml' repository path.
- The '--pyproject-requirements' option overrides the 'pyproject.toml' dependencies list.
- The '--pyproject-skip-metapaths' option overrides the 'pyproject.toml' skip metapath setting.
- The '--pyproject-exclude' option overrides the 'pyproject.toml' exclude file path.
- The '--pyproject-include' option overrides the 'pyproject.toml' include file path.
- The '--pyproject-skip-artifacts' option overrides the 'pyproject.toml' skip artifacts setting.
- The '--pyproject-exclude-include' option overrides the 'pyproject.toml' exclude/include 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	70 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414-415, 417-418, 420-421, 424-426, 428, 430, 432, 434-436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the test_load_from_cli_provider_override method correctly overrides the pyproject.toml configuration.

Why Needed: This test prevents a potential bug where the CLI provider option is not overridden in the pyproject.toml file, potentially leading to unexpected behavior or errors during the loading of dependencies.

Key Assertions:

- The 'pyproject' key in the pyproject.toml file should be set to 'override' when the CLI provider option is used.
- The value of the 'pyproject' key in the pyproject.toml file should match the expected override value.
- The test should fail if the 'pyproject' key in the pyproject.toml file does not contain the word 'override'.
- The test should succeed if the 'pyproject' key in the pyproject.toml file contains the word 'override'.
- The 'pyproject' key in the pyproject.toml file should be set to a value other than 'override' when the CLI provider option is used.
- The value of the 'pyproject' key in the pyproject.toml file should not match the expected override value when the CLI provider option is used.
- The test should fail if the 'pyproject' key in the pyproject.toml file does not contain a valid override value (e.g., 'override', 'override=')
- The test should succeed if the 'pyproject' key in the pyproject.toml file contains a valid override value (e.g., 'override', 'override=')

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	68 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414-415, 417-418, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the `llm_max_retries` option is correctly set to 2 when loading configuration from CLI.

Why Needed: This test prevents a potential bug where the `llm_max_retries` option is not properly set, leading to incorrect configuration.

Key Assertions:

- The value of `llm_max_retries` should be 2.
- The `load_config` function should return an instance with `llm_max_retries` set to 2.
- The `cfg` variable should contain a dictionary with the correct key-value pair for `llm_max_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/options.py	26 lines (ranges: 123, 163, 276, 279-280, 288-290, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436-437, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM 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	69 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-304, 308, 310, 312, 316, 320, 324, 328-330, 332, 334, 336, 340, 342, 346, 348, 350-352, 354-356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_options_coverage.py

38 tests

PASSED `tests/test_options_coverage.py::TestCliOverrides::test_cli_dependency_snapshot` 2ms 3

AI ASSESSMENT

Scenario: Verify that the `llm_dependency_snapshot` option is correctly set to 'deps.json' when running CLI with overrides.

Why Needed: This test prevents a potential bug where the `llm_dependency_snapshot` option is not set correctly, potentially leading to incorrect dependency snapshots being reported.

Key Assertions:

- The value of `llm_dependency_snapshot` in the configuration file should be 'deps.json'.
- The `report_dependency_snapshot` key should contain the string 'deps.json'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	29 lines (ranges: 123, 163, 276, 279-280, 288-290, 414-415, 417-418, 420-421, 424, 426, 428, 430, 432-434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestCliOverrides::test_cli_evidence_bundle` 2ms 3

AI ASSESSMENT

Scenario: Verify that the `test_cli_evidence_bundle` test case correctly sets the `llm_evidence_bundle` option to 'bundle.zip' in the configuration.

Why Needed: This test prevents a potential bug where the `llm_evidence_bundle` option is not set correctly, potentially leading to incorrect report generation or other issues.

Key Assertions:

- The `llm_evidence_bundle` option is set to 'bundle.zip' in the configuration.
- The `report_evidence_bundle` value matches 'bundle.zip'.
- The `llm_evidence_bundle` option has a valid value ('bundle.zip').
- The configuration file contains the correct `llm_evidence_bundle` option value ('bundle.zip').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	29 lines (ranges: 123, 163, 276, 279-280, 288-290, 414-415, 417-418, 420-421, 424, 426, 428, 430-432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestCliOverrides::test_cli_report_json` 2ms 3

AI ASSESSMENT

Scenario: Verifies that the test_cli_report_json function sets the report JSON option to 'output.json' when CLI override is enabled.

Why Needed: This test prevents a bug where the report JSON option is not set correctly when CLI overrides are enabled.

Key Assertions:

- mock.option.llm_report_json = 'output.json'
- cfg.report_json == 'output.json'
- assert cfg._report_json == 'output.json'
- self._test_cli_report_json(tmp_path) should have been called with the correct config
- mock._report_json is set to 'output.json' when CLI override is enabled

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	29 lines (ranges: 123, 163, 276, 279-280, 288-290, 414-415, 417-418, 420-421, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestCliOverrides::test_cli_report_pdf` 2ms 3

AI ASSESSMENT

Scenario: Verifies that the `test_cli_report_pdf` test function sets the `llm_report_pdf` option to 'output.pdf' in the mock configuration.

Why Needed: This test prevents a regression where the `llm_report_pdf` option is not set correctly when running the CLI with overrides.

Key Assertions:

- The value of `llm_report_pdf` is set to 'output.pdf'.
- The value of `llm_report_pdf` matches the expected value in the mock configuration.
- The `llm_report_pdf` option is present in the mock configuration.
- The `llm_report_pdf` option has a value of 'output.pdf' in the mock configuration.
- The `llm_report_pdf` option does not have any other values in the mock configuration.
- The `llm_report_pdf` option is set to 'output.pdf' when the test runs with overrides.
- The `llm_report_pdf` option is set to 'output.pdf' in the mock configuration when the test runs without overrides.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	29 lines (ranges: 123, 163, 276, 279-280, 288-290, 414-415, 417-418, 420-421, 424, 426, 428-430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestConfigValidationCoverage::test_validate_invalid_token_output_format` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `validate()` method of the `Config` class returns an error when the token output format is invalid.

Why Needed: This test prevents a potential bug where the `validate()` method does not raise an error for invalid token output formats, potentially leading to unexpected behavior or errors in downstream code.

Key Assertions:

- The `validate()` method of the `Config` class should return an error when the token output format is specified as 'xml'.
- Any error messages returned by the `validate()` method should contain the string 'litellm_token_output_format' to ensure a clear indication of invalid input.
- The test should fail if no error message containing 'litellm_token_output_format' is found in the errors list.
- The `validate()` method should raise an exception when called with an invalid token output format, indicating that the input was incorrect.
- Any exceptions raised by the `validate()` method should be caught and reported as an error, preventing the test from passing even if the input is valid.
- The test should only fail if the input is not a string or None, to prevent false positives due to other types of errors.
- The `validate()` method should raise a `ValueError` exception when called with an invalid token output format, indicating that the input was incorrect.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	23 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-229, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestConfigValidationCoverage::test_validate_token_refresh_interval_too_short` 1ms 3

AI ASSESSMENT

Scenario: Test validation when token refresh interval is too short.

Why Needed: Prevents a potential bug where the token refresh interval is set to a value that may cause issues with the application's security and functionality.

Key Assertions:

- The `litellm_token_refresh_interval` must be at least 60.
- Any invalid values for `litellm_token_refresh_interval` will result in an error message indicating that it should be at least 60.

COVERAGE

src/pytest_llm_report/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: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233-234, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestConfigValidationCoverage::test_validate_valid_litellm_config` 1ms 3

AI ASSESSMENT

Scenario: Test validation of valid LiteLLM configuration.

Why Needed: Prevents regression where invalid or outdated config is used with LitLLM.

Key Assertions:

- The `validate()` method returns an empty list of errors if the provided configuration is valid.
- The `litellm_token_output_format` parameter is set to 'text' and the `litellm_token_refresh_interval` parameter is set to 3600 seconds.
- The `provider` parameter is set to 'litellm'.
- The `token_output_format` parameter is not provided. It should be either 'text', 'json', or 'csv'.
- The `refresh_interval` parameter is not a valid value for LitLLM. It can only take values of 60, 300, or 3600 seconds.
- The `provider` and `token_output_format` parameters are both 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/options.py	20 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_aggregate_include_history` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `aggregate_include_history` option is loaded correctly from the `pyproject.toml` file.

Why Needed: This test prevents a potential bug where the `aggregate_include_history` option is not loaded, potentially causing issues with coverage analysis.

Key Assertions:

- The `aggregate_include_history` option is present in the `pyproject.toml` file.
- The `aggregate_include_history` option has the correct value ('include' or 'exclude').
- The `include` and `exclude` values are correctly formatted (e.g., 'include=True', 'exclude=False').
- The `include` and `exclude` values do not contain any invalid characters (e.g., spaces, special characters).
- The `aggregate_include_history` option is loaded from the correct location in the `pyproject.toml` file.
- The `aggregate_include_history` option is correctly included in the coverage 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`	66 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392-394, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_aggregate_policy_from_pyproject` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `load_aggregate_policy` function can successfully load an aggregate policy from a PyProject file.

Why Needed: This test prevents a potential bug where the `load_aggregate_policy` function fails to load aggregate policies from PyProjects due to incorrect or missing configuration files.

Key Assertions:

- The `load_aggregate_policy` function should be able to read and parse the aggregate policy from the provided PyProject file.
- The `load_aggregate_policy` function should raise a `FileNotFoundError` if the specified PyProject file is not found or does not exist.
- The `load_aggregate_policy` function should correctly load the aggregate policy configuration from the PyProject file, including any required dependencies and settings.
- The `load_aggregate_policy` function should be able to handle cases where the PyProject file contains invalid or incomplete configuration data.
- The `load_aggregate_policy` function should not raise an exception if the PyProject file is empty or does not contain any aggregate policy information.
- The `load_aggregate_policy` function should correctly load the aggregate policy from a PyProject file that uses a custom configuration file (e.g. `.yml` or `.json`).
- The `load_aggregate_policy` function should be able to handle cases where the PyProject file is located in a non-standard location (e.g. a directory instead of a file).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390-392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_all_config_keys_combined` 2ms 3

AI ASSESSMENT

Scenario: Verify that the 'load_all_config_keys' function loads all required configuration keys from a single pyproject.toml file.

Why Needed: This test prevents a potential bug where the 'load_all_config_keys' function does not load all required configuration keys, potentially leading to coverage issues or unexpected behavior.

Key Assertions:

- The function should be able to read and write the 'pyproject.toml' file successfully without any errors.
- All required configuration keys should be loaded from the pyproject.toml file.
- The function should return an empty dictionary if no configuration keys are found in the pyproject.toml file.
- The function should raise a ValueError if the pyproject.toml file is missing or corrupted.
- The function should not throw any exceptions when writing to the pyproject.toml file.
- All required configuration keys should be present in the 'pyproject.toml' file with the correct data types and values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	106 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-305, 308-314, 316-318, 320-322, 324-325, 328-337, 340-343, 346-359, 362-364, 366-367, 370-376, 380-382, 384-386, 390-394, 398-401, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_cache_dir` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `load_cache_dir` function loads the correct cache directory from the pyproject.toml file.

Why Needed: This test prevents a potential bug where the cache directory is not loaded correctly, potentially leading to issues with caching and dependency resolution.

Key Assertions:

- The `pyproject.toml` file contains the correct path to the cache directory.
- The `load_cache_dir` function writes the correct path to the cache directory to the pyproject.toml file.
- The `cache_dir` attribute of the `Project` object is set correctly to the loaded cache directory.
- The `pyproject.toml` file contains a valid `cache_dir` attribute with the expected value.
- The `load_cache_dir` function does not raise an exception when the cache directory cannot be loaded.
- The `cache_dir` attribute of the `Project` object is set correctly to the cached value.
- The `pyproject.toml` file contains a valid `cache_dir` attribute with the expected value.
- The `load_cache_dir` function does not raise an exception when the cache directory cannot be loaded.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358-359, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_cache_ttl_seconds` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `cache_ttl_seconds` value is loaded correctly from `pyproject.toml`.

Why Needed: This test prevents a potential bug where the `cache_ttl_seconds` value is not properly loaded, potentially leading to incorrect cache behavior.

Key Assertions:

- The value of `cache_ttl_seconds` in `pyproject.toml` matches the expected value.
- The `cache_ttl_seconds` value is correctly formatted and does not contain any invalid characters.
- The `cache_ttl_seconds` value is greater than or equal to 0 seconds.
- The `cache_ttl_seconds` value is less than the maximum allowed value (3600 seconds).
- The file path of `pyproject.toml` exists and is accessible.
- The test runs without any errors or exceptions.
- The cache TTL seconds are correctly set for a given 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	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356-358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_capture_failed_output` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `load_capture_failed_output` function can successfully load the `capture_failed_output` setting from the `pyproject.toml` file.

Why Needed: This test prevents a regression where the `load_capture_failed_output` function fails to load the `capture_failed_output` setting due to an incorrect or missing configuration in the `pyproject.toml` file.

Key Assertions:

- The `pyproject.toml` file should contain the correct value for the `capture_failed_output` setting.
- The `load_capture_failed_output` function should be able to successfully load this value from the `pyproject.toml` file.
- The `pyproject.toml` file should not be missing or empty, causing the `load_capture_failed_output` function to fail.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372-374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_capture_output_max_chars` 1ms 3

AI ASSESSMENT

Scenario: Verify that `load_capture_output_max_chars` loads the correct value from `pyproject.toml`.

Why Needed: This test prevents a potential bug where the `max_chars` value is not correctly loaded due to an incorrect or missing configuration in the `pyproject.toml` file.

Key Assertions:

- The value of `load_capture_output_max_chars` is equal to the expected value from `pyproject.toml`.
- The value of `max_chars` is a valid integer between 1 and 1000 (inclusive).
- The configuration in `pyproject.toml` is correctly formatted and does not contain any invalid characters or syntax.
- The test output is as expected, indicating that the `load_capture_output_max_chars` function loaded the correct value from `pyproject.toml`.
- The value of `max_chars` is a multiple of 10 (e.g., 100, 200, etc.) to ensure accurate capture output.
- No exceptions are raised during the execution of the test, indicating that the `load_capture_output_max_chars` function is working 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	66 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374-376, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_context_bytes` 1ms 3

AI ASSESSMENT

LLM 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	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330-332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_context_exclude_globs` 1ms 3

AI ASSESSMENT

Scenario: Verify the `load_context_exclude_globs` function in `test_load_context_exclude_globs` loads context_exclude_globs correctly from `pyproject.toml`.

Why Needed: This test prevents a potential bug where the `load_context_exclude_globs` function fails to load context_exclude_globs from pyproject.toml due to incorrect or missing file paths.

Key Assertions:

- The `load_context_exclude_globs` function writes the correct value of `context_exclude_globs` to `pyproject.toml`.
- The `load_context_exclude_globs` function correctly handles cases where `context_exclude_globs` is not found in pyproject.toml.
- The `load_context_exclude_globs` function does not throw any errors when the file path is incorrect or missing.
- The `load_context_exclude_globs` function does not throw any errors when the value of `context_exclude_globs` is empty.
- The `load_context_exclude_globs` function correctly handles cases where `context_exclude_globs` has a different format than expected (e.g., multiple values separated by commas).
- The `load_context_exclude_globs` function does not throw any errors when the file path or value of `context_exclude_globs` is not a string.
- The `load_context_exclude_globs` function correctly handles cases where `context_exclude_globs` has a different encoding than expected (e.g., UTF-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/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336-337, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_context_file_limit` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `context_file_limit` setting is correctly loaded from pyproject.toml.

Why Needed: This test prevents a potential bug where the `context_file_limit` setting is not properly loaded, potentially causing issues with context loading in certain scenarios.

Key Assertions:

- The content of the `pyproject.toml` file matches the expected format.
- The `tmp_path / 'pyproject.toml'` path is correctly constructed and resolves to a valid file.
- The `pyproject.write_text()` method writes the correct text to the file.
- The `context_file_limit` setting is not set in the `pyproject.toml` file.
- The `tmp_path / 'pyproject.toml'` path does not resolve to an existing file.
- The `pyproject.toml` file has a valid and correct format.
- The test fails with an error message indicating that the setting was not found in the file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332-334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_context_include_globs` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `context_include_globs` setting is loaded correctly from pyproject.toml.

Why Needed: This test prevents a potential bug where the `context_include_globs` setting is not loaded, potentially leading to incorrect coverage analysis.

Key Assertions:

- The contents of `pyproject.toml` are checked for the presence of `context_include_globs`.
- The value of `context_include_globs` in `pyproject.toml` is verified to be set correctly.
- The `context_include_globs` setting is loaded from `pyproject.toml` and used during testing.
- The coverage analysis includes the specified globs when running tests with the `--context` flag.
- The test environment has access to the loaded `context_include_globs` setting.
- The test output shows that the globs are correctly included in the coverage report.
- The test passes without any errors or warnings related to the `context_include_globs` setting.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334-336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_hmac_key_file` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `hmac_key_file` is loaded correctly from a PyProject file.

Why Needed: This test prevents a potential issue where the `hmac_key_file` is not loaded, potentially causing coverage issues or incorrect behavior in downstream code.

Key Assertions:

- The contents of the `pyproject.toml` file are correct and match the expected values.
- The `hmac_key_file` is present in the PyProject file and is correctly referenced.
- The `hmac_key_file` is loaded successfully without any errors or warnings.
- The coverage of the code using the `hmac_key_file` is sufficient to ensure correct behavior.
- The `pyproject.toml` file has the correct `hmac_key_file` value.
- The `hmac_key_file` is not empty and contains valid HMAC key data.
- The `hmac_key_file` is loaded correctly from a PyProject file with an existing `hmac_key_file` 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	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400-401, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_include_param_values` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `include_param_values` option is correctly loaded from pyproject.toml.

Why Needed: This test prevents a potential bug where the `include_param_values` option is not properly loaded, potentially causing issues with code coverage.

Key Assertions:

- The contents of the `pyproject.toml` file are correct and match the expected values.
- The `include_param_values` option is correctly set to `True` in the pyproject.toml file.
- The `include_param_values` option is not set to `False` or any other value in the pyproject.toml file.
- The contents of the included files are correctly loaded and available for use in the code.
- The test environment has the necessary permissions to read the pyproject.toml file.
- The `include_param_values` option is not enabled by default in the project settings.
- The `include_param_values` option is properly configured in the project's settings file (if applicable).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340-342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_include_phase` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `include_phase` is loaded correctly from the PyProject.toml file.

Why Needed: This test prevents a potential bug where the include phase is not loaded, potentially leading to incorrect coverage analysis.

Key Assertions:

- The contents of the `include_phase` section in the PyProject.toml file are correct.
- The `include_phase` section is present and contains the expected values.
- The `include_phase` section does not contain any invalid or empty lines.
- The `include_phase` section has the correct indentation and formatting.
- The `include_phase` section includes all required files and directories.
- The `include_phase` section excludes unnecessary files and directories.
- All required dependencies are included in the `include_phase` section.
- No empty lines or whitespace are present in the `include_phase` section.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366-367, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_include_pytest_invocation` 1ms 3

AI ASSESSMENT

LLM 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	66 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380-382, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_invocation_redact_patterns` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `load_invocation_redact_patterns` function can successfully load invocation_redact_patterns from a PyProject.toml file.

Why Needed: This test prevents a potential bug where the `load_invocation_redact_patterns` function fails to load invocation_redact_patterns due to a missing or corrupted PyProject.toml file.

Key Assertions:

- The function should be able to read the 'invocation_redact_patterns' key from the PyProject.toml file without raising an exception.
- The value of the 'invocation_redact_patterns' key should match the expected pattern.
- The function should raise a `FileNotFoundError` if the PyProject.toml file does not exist or is missing the required keys.
- The function should raise a `KeyError` if the 'invocation_redact_patterns' key is not found in the PyProject.toml file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	66 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384-386, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_litellm_api_base` 1ms 3

AI ASSESSMENT

Scenario: The 'litellm_api_base' module should be loaded successfully when the test is run.

Why Needed: This test prevents a potential issue where the 'litellm_api_base' module might not be loaded due to a missing or corrupted pyproject.toml file.

Key Assertions:

- The contents of pyproject.toml are correct and do not contain any errors.
- The 'pyproject.toml' file is successfully written to the specified path.
- The 'litellm_api_base' module can be loaded without any issues when the test is run.
- No exceptions are raised during the loading process of the 'litellm_api_base' module.
- The 'pyproject.toml' file is correctly formatted and does not contain any invalid characters or syntax 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	66 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-302, 304, 308-310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_litellm_api_key` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `litellm_api_key` is loaded correctly from the `pyproject.toml` file.

Why Needed: This test prevents a potential bug where the API key is not loaded due to an issue with the `pyproject.toml` file being corrupted or incomplete.

Key Assertions:

- The contents of the `pyproject.toml` file should contain the expected `litellm_api_key` value.
- The `pyproject.toml` file should be able to read and write the `litellm_api_key` value correctly.
- The test should fail if the `pyproject.toml` file is empty or missing the required `litellm_api_key` key.
- The `litellm_api_key` value should be a string containing only alphanumeric characters and underscores.
- The `litellm_api_key` value should not contain any whitespace characters.
- The test should pass if the `pyproject.toml` file contains the expected `litellm_api_key` value in the correct 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/options.py	66 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-302, 304, 308, 310-312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_litellm_token_json_key` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `litellm_token_json_key` key is correctly loaded from the `pyproject.toml` file.

Why Needed: This test prevents a potential issue where the `litellm_token_json_key` value is not properly set or is missing in the `pyproject.toml` file, potentially leading to coverage issues.

Key Assertions:

- The 'litellm_token_json_key' key should be present and have the correct value.
- The value of 'litellm_token_json_key' should match the expected value from the `litellm_token.json` file.
- The 'litellm_token_json_key' value should not be empty or null.
- The 'litellm_token_json_key' value should be a valid JSON string.
- The 'litellm_token_json_key' value should match the expected format (e.g., 'key: value').
- The 'pyproject.toml' file should contain the necessary configuration for litellm to load the token JSON key 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	66 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-302, 304, 308, 310, 312, 316, 320, 324-325, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_litellm_token_output_format` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `litellm_token_output_format` option in the `pyproject.toml` file is correctly loaded and used to generate token output.

Why Needed: This test prevents a potential bug where the `litellm_token_output_format` option is not properly loaded or configured, resulting in incorrect or missing token output.

Key Assertions:

- The `pyproject.toml` file should contain a section named `litellm.token_output_format` with a value of either 'json' or 'txt'.
- The `pyproject.toml` file should contain the correct path to the `litellm_token_output_format` function.
- The `pyproject.toml` file should contain the correct arguments for the `litellm_token_output_format` function (e.g., `output_path`, `format`)
- The generated token output should be in the expected format (either 'json' or 'txt').
- The test should fail when the `pyproject.toml` file is missing the `litellm.token_output_format` option or has an incorrect value.
- The test should pass when the `pyproject.toml` file contains the correct `litellm.token_output_format` option and 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/options.py	67 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-302, 304, 308, 310, 312, 316, 320-322, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_litellm_token_refresh_command` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `litellm_token_refresh_command` command is loaded correctly from the PyProject.toml file.

Why Needed: This test prevents a potential bug where the `litellm_token_refresh_command` command is not loaded, potentially causing issues with the application's functionality or behavior.

Key Assertions:

- The contents of the `pyproject.toml` file are correct and match the expected values.
- The `litellm_token_refresh_command` key exists in the PyProject.toml file and its value is set to a valid string.
- The command's name, 'refresh', is present in the list of commands defined in the PyProject.toml file.
- The command's description, 'Refresh litellm token', is present in the list of descriptions defined in the PyProject.toml file.
- The command's author and license information are correctly set in the PyProject.toml file.
- The `litellm_token_refresh_command` command has a valid path to the required files.
- The command's dependencies are correctly specified in the PyProject.toml file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	67 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-302, 304, 308, 310, 312-314, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_litellm_token_refresh_interval` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `litellm_token_refresh_interval` option is loaded correctly from the `pyproject.toml` file.

Why Needed: This test prevents a potential bug where the `litellm_token_refresh_interval` option is not loaded, potentially leading to coverage issues or unexpected behavior in the application.

Key Assertions:

- The contents of the `pyproject.toml` file are correct and match the expected values.
- The `litellm_token_refresh_interval` option is present and has the correct value (e.g., 1 day, 7 days, etc.).
- The `pyproject.toml` file does not contain any other options that could potentially prevent loading of `litellm_token_refresh_interval`.
- The test loads the `pyproject.toml` file successfully without encountering any errors or exceptions.
- The value of `litellm_token_refresh_interval` is correctly converted to a string (e.g., '1d', '7d', 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/options.py	67 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-302, 304, 308, 310, 312, 316-318, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_malformed_pyproject` 1ms 3

AI ASSESSMENT

Scenario: Test loading of malformed pyproject.toml file.

Why Needed: This test prevents a potential bug where the 'provider' is set to 'none' when a malformed pyproject.toml is encountered, causing the default provider to be used instead.

Key Assertions:

- The 'cfg.provider' attribute should not be set to 'none'.
- The 'cfg.provider' attribute should be set to 'default' or an empty string if no valid configuration is found.
- When a malformed pyproject.toml file is encountered, the test should fallback to defaults and not raise an exception.
- The default provider ('default') should be used when the malformed config cannot be parsed 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	34 lines (ranges: 123, 163, 276, 279-280, 288-293, 403, 405, 407-410, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_max_concurrency` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `max_concurrency` setting in a Pyproject.toml file is correctly loaded.

Why Needed: This test prevents a potential bug where the `max_concurrency` value is not properly loaded from the Pyproject.toml file, potentially leading to unexpected behavior or errors when using the `concurrent.futures.ThreadPoolExecutor` class.

Key Assertions:

- The contents of the `pyproject.toml` file are correctly read and parsed as a JSON object.
- The `max_concurrency` value is present in the `pyproject.toml` file and is not empty or zero.
- The `max_concurrency` value is an integer.
- The `max_concurrency` value does not exceed 1000 (the maximum allowed value).
- The `concurrent.futures.ThreadPoolExecutor` class can correctly use the specified concurrency level.
- The test fails when the `max_concurrency` value exceeds 1000, indicating a bug in Pyproject.toml loading.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348-350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_max_tests` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `max_tests` setting in the `pyproject.toml` file is correctly loaded.

Why Needed: This test prevents a potential bug where the maximum number of tests is not properly set, potentially leading to coverage issues.

Key Assertions:

- The contents of the `pyproject.toml` file are correct and match the expected values.
- The `max_tests` setting in the `pyproject.toml` file is correctly loaded into the test environment.
- The number of tests loaded matches the maximum value specified in the `pyproject.toml` file.
- No coverage issues are reported when running the tests with the correct maximum number of tests.
- The test environment has the required configuration for loading the maximum number of tests.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346-348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_metadata_file` 1ms 3

AI ASSESSMENT

LLM 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	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398-400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_ollama_host` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `ollama_host` is loaded correctly from the `pyproject.toml` file.

Why Needed: This test prevents a potential issue where the `ollama_host` is not loaded due to a missing or corrupted `pyproject.toml` file.

Key Assertions:

- The contents of the `pyproject.toml` file are correct and match the expected values.
- The `pyproject.toml` file exists at the specified path.
- The `ollama_host` is present in the `pyproject.toml` file with the correct value.
- The `ollama_host` value matches the expected value from the `ollama.yaml` file.
- The `pyproject.toml` file is not corrupted or missing any necessary files.
- The `pyproject.toml` file is written to the specified path without any issues.
- The `ollama_host` value in the `pyproject.toml` file matches the expected value from the `ollama.yaml` file.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`66 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-302, 304-305, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_omit_tests_from_coverage` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `load_omit_tests_from_coverage` function in `tests/test_options_coverage.py` can be loaded successfully from a PyProject.toml file.

Why Needed: This test prevents a potential bug where the `load_omit_tests_from_coverage` function is not able to load correctly from a PyProject.toml file, potentially causing coverage issues.

Key Assertions:

- The `load_omit_tests_from_coverage` function should be able to read and write data from the PyProject.toml file successfully.
- The `load_omit_tests_from_coverage` function should not raise any exceptions when loading from a valid PyProject.toml file.
- The `load_omit_tests_from_coverage` function should correctly omit tests from coverage reports for the specified configuration.
- The `load_omit_tests_from_coverage` function should handle invalid or missing PyProject.toml files without raising an exception.
- The `load_omit_tests_from_coverage` function should not modify any existing coverage reports.
- The `load_omit_tests_from_coverage` function should preserve the original configuration settings in case of a failure to load.
- The `load_omit_tests_from_coverage` function should log any errors that occur during loading, if necessary.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	66 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362-364, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_param_value_max_chars` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `param_value_max_chars` value is loaded correctly from the PyProject.toml file.

Why Needed: This test prevents a potential bug where the `param_value_max_chars` value is not loaded, potentially leading to coverage issues or incorrect results.

Key Assertions:

- The content of the 'pyproject.toml' file matches the expected string.
- The 'max_chars' attribute of the 'param_value_max_chars' object in the PyProject.toml file matches the expected value.
- The 'value' attribute of the 'param_value_max_chars' object in the PyProject.toml file contains the correct maximum character count.
- The 'type' attribute of the 'param_value_max_chars' object in the PyProject.toml file is set to 'str'.
- The 'default' value of the 'max_chars' attribute of the 'param_value_max_chars' object in the PyProject.toml file is not set to a default value.
- The 'default' value of the 'max_chars' attribute of the 'param_value_max_chars' object in the PyProject.toml file is less than or equal to 100 characters.
- The 'max_chars' attribute of the 'param_value_max_chars' object in the PyProject.toml file contains a string with fewer than 100 characters.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342-343, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_report_collect_only` 1ms 3

AI ASSESSMENT

LLM 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	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370-372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_coverage.py::TestPyprojectLoadingCoverage::test_load_timeout_seconds` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `timeout_seconds` value in `pyproject.toml` is correctly loaded when the test runs for a specified number of seconds.

Why Needed: This test prevents a potential regression where the timeout value is not updated correctly if the test runs for an extended period.

Key Assertions:

- The `timeout_seconds` value in `pyproject.toml` is written to disk and can be read back by the test.
- The `timeout_seconds` value is set to a non-zero value when the test starts and remains unchanged even after the test runs for an extended period.
- The `timeout_seconds` value is correctly updated when the test runs for a specified number of seconds (e.g., 5 seconds).
- The `timeout_seconds` value is not updated if the test runs for less than 5 seconds.
- The `timeout_seconds` value in `pyproject.toml` can be read back even after the test has finished running.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	65 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300, 302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352-354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_options_extended.py

13 tests

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

AI ASSESSMENT

Scenario: Test Config with aggregation settings.

Why Needed: Prevents regression in aggregation settings configuration.

Key Assertions:

- The `aggregate_dir` attribute is set to `/reports`.
- The `aggregate_policy` attribute is set to 'merge'.
- The `aggregate_include_history` attribute is set to True.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 123, 163)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that all output paths are set correctly in the test configuration.

Why Needed: This test prevents a potential bug where the report generation process fails to produce expected output files due to missing or incorrect input configurations.

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: 123, 163)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that `capture_failed_output` is set to `True`.

Why Needed: This test prevents a potential issue where the captured output exceeds the maximum allowed length of 8000 characters.

Key Assertions:

- config.capture_failed_output is True
- config.capture_output_max_chars is 8000

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the `Config` object correctly sets its metadata file and HMAC key file.

Why Needed: This test prevents a potential bug where the configuration is not set with the expected files.

Key Assertions:

- The `metadata_file` attribute of the `Config` object should be set to 'metadata.json'.
- The `hmac_key_file` attribute of the `Config` object should be 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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 regression where the test coverage settings are not correctly configured.

Key Assertions:

- config.omit_tests_from_coverage is set to False
- config.include_phase is set to "all"

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the `llm_context_exclude_globs` attribute of the `Config` class is correctly populated with custom exclude globs.

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

Key Assertions:

- The `*.pyc` glob should be present in the `llm_context_exclude_globs` list.
- The `*.log` glob should be present in the `llm_context_exclude_globs` list.
- All custom exclude globs should 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: 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the `include_globs` option includes only `.py` files.

Why Needed: This test prevents a potential bug where the `include_globs` option is not correctly applied, potentially leading to unexpected behavior or errors.

Key Assertions:

- The `*.py` glob matches any file with a `.py` extension in the specified directory.
- The `*.pyi` glob matches any file with an `.pyi` extension in the specified directory.
- The `include_globs` option includes only files that match either of these patterns.
- If the `include_globs` option does not include all required files, it will fail to compile or run correctly.
- Including non-`.py` and `*.pyi` files in the `include_globs` list can cause issues with LLMs.

COVERAGE

src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 123)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the LLM execution settings are correctly configured.

Why Needed: This test prevents a potential regression where the LLM execution settings are not properly set, potentially leading to performance issues or errors.

Key Assertions:

- The value of llm_max_tests is indeed 50.
- The value of llm_max_concurrency is indeed 8.
- The value of llm_requests_per_minute is indeed 12.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Configures the LLM with param settings.

Why Needed: Prevents a potential bug where the LLM parameter values exceed the maximum allowed length.

Key Assertions:

- config.llm_include_param_values is True
- config.llm_param_value_max_chars == 200

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test the configuration of LLM settings for the OLLAMA provider.

Why Needed: Prevents a potential bug where the model and context bytes are not correctly configured when using the OLLAMA provider.

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 64KB (64000).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the `repo_root` attribute of a `TestConfigAnnotations` instance is correctly set to the specified path.

Why Needed: This test prevents a potential bug where the `repo_root` attribute is not set correctly, potentially leading to incorrect configuration or unexpected behavior.

Key Assertions:

- config.repo_root is equal to Path('/project')
- config.repo_root is an instance of `path.Path`
- config.repo_root is a valid path
- config.repo_root is not empty
- config.repo_root does not start with a slash
- config.repo_root is not relative
- config.repo_root is not absolute

COVERAGE

src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test the `test_valid_phase_values` function to ensure it validates include_phase values correctly.

Why Needed: This test prevents potential issues where invalid or missing include_phase values could cause validation failures.

Key Assertions:

- All 'run', 'setup', and 'teardown' phases should be included in the configuration without any errors.
- The 'all' phase should not have any errors if it is valid.
- Any invalid or missing include_phase value should be excluded from validation results.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_options_maximal.py

10 tests

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

AI ASSESSMENT

Scenario: Verify that the default exclude globs returned by `Config().llm_context_exclude_globs` match the expected values.

Why Needed: This test prevents a potential bug where the default exclude globs are not correctly set, potentially leading to incorrect analysis or errors in the model.

Key Assertions:

- The function `*.pyc` should be included in the default exclude globs.
- The function `__pycache__/*` should also be included in the default exclude globs.
- The string `*secret*` should be included in the default exclude globs.
- The string `*password*` should be included in the default exclude globs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Test default redact patterns with various configuration options.

Why Needed: Prevents a potential bug where the default redact patterns do not include sensitive information like passwords and tokens, potentially exposing user credentials or API keys.

Key Assertions:

- The `--password` pattern should be present in the default redact patterns.
- The `--token` pattern should be present in the default redact patterns.
- The `--api[_-]?key` pattern should be present in the default redact patterns.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: The test verifies that the default values of the `get_default_config()` function are correct.

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

Key Assertions:

- config.provider == 'none'
- config.llm_context_mode == 'minimal'
- config.llm_context_bytes == 32000
- config.omit_tests_from_coverage is True
- config.include_phase == 'run'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_maximal.py::TestConfigHelpersMaximal::test_is_llm_enabled` 1ms 3

AI ASSESSMENT

Scenario: Verifies that the `is_llm_enabled` method returns False for a provider without an LLM, and True for providers with LLMs.

Why Needed: Prevents regression in case a new provider is added without an LLM.

Key Assertions:

- The `is_llm_enabled` method should return `False` when the provider has no known LLM.
- The `is_llm_enabled` method should return `True` when the provider has an LLM (e.g., 'ollama', 'litellm', or 'gemini').

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`3 lines (ranges: 123, 163, 252)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_aggregate_policy` 1ms 3

AI ASSESSMENT

Scenario: Test the validation of an invalid aggregate policy.

Why Needed: To prevent a potential bug where an invalid aggregate policy is passed to the Config class, causing unexpected behavior or errors.

Key Assertions:

- The configuration object should have exactly one error message.
- The error message should contain 'Invalid aggregate_policy 'invalid''
- The error message should be present in the first error message of the list.
- The error message should not be empty.
- The error message should not be a string.
- The error message should not be a file path or any other type of object.
- The configuration object should have been validated successfully before 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/options.py	23 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-213, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_context_mode` 1ms 3

AI ASSESSMENT

LLM 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	23 lines (ranges: 123, 163, 191, 194-195, 201-205, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_include_phase` 1ms 3

AI ASSESSMENT

Scenario: Test the `validate` method of the `Config` class to ensure it correctly identifies and reports invalid include phases.

Why Needed: Prevents a potential bug where an invalid include phase is silently ignored, potentially leading to incorrect configuration or errors in downstream code.

Key Assertions:

- The `validate` method returns exactly one error for the given invalid include phase.
- The error message contains the string 'Invalid include_phase 'invalid' which indicates the specific issue with the input value.
- The test asserts that at least one error is found in the list of errors returned by the `validate` method.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	23 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-221, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_provider` 1ms 3

AI ASSESSMENT

LLM 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	22 lines (ranges: 123, 163, 191, 194-197, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_numeric_bounds` 1ms 3

AI ASSESSMENT

Scenario: Should return errors for invalid numeric values.

Why Needed: This test prevents a potential bug where the Config class does not validate its inputs correctly, potentially leading to unexpected behavior or errors when using the config.

Key Assertions:

- llm_context_bytes must be a non-negative integer.
- llm_max_tests must be an integer greater than 0.
- llm_requests_per_minute must be an integer greater than 0.
- llm_timeout_seconds must be a positive integer.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	24 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237-245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_valid_config` 1ms 3

AI ASSESSMENT

Scenario: Tests the `validate` method of the `Config` class with a valid configuration.

Why Needed: Prevents potential infinite recursion or other unexpected behavior in case of an invalid configuration.

Key Assertions:

- The `validate()` method should return an empty list for a well-formed, valid configuration.
- The `validate()` method should not throw any exceptions for a valid configuration.
- The method should correctly handle cases where the input is malformed or inconsistent.
- The method should not attempt to perform any actions on the input that are not valid for the given configuration.
- Any potential side effects of the `validate()` method should be properly cleaned up after use.

COVERAGE

`src/pytest_llm_report/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: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

📄 tests/test_plugin_integration.py

14 tests

PASSED tests/test_plugin_integration.py::TestPluginConfigLoading::test_config_defaults 3ms 3

AI ASSESSMENT

Scenario: Test that the config defaults to safe settings when no options are provided.

Why Needed: Prevents a potential bug where the config is set to unexpected or insecure values without explicit configuration.

Key Assertions:

- The `cfg` variable should be an instance of `Config`.
- The `cfg` variable should not have any explicitly registered options.
- The `cfg` variable should have safe defaults for all settings.
- The `cfg` variable should not contain any sensitive or insecure 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	80 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-305, 308, 310, 312, 316, 320, 324, 328-330, 332, 334, 336, 340, 342, 346-348, 350, 352-354, 356, 358, 362-364, 366-367, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414-415, 417-418, 420-421, 424-430, 432, 434, 436, 440, 442, 444-446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_plugin_integration.py::TestPluginConfigLoading::test_markers_exist_in_config` 1ms 2

AI ASSESSMENT

Scenario: Verify that the `pytestconfig` object exists and is not None.

Why Needed: Prevent a potential bug where the plugin configuration is inaccessible due to a missing or corrupted configuration.

Key Assertions:

- assert pytestconfig is not None
- assert isinstance(pytestconfig, pytest.Config)
- pytest.config.dictConfig() should return a dictionary-like object
- pytest.config.getopt_list() should return an empty list if no options are provided
- pytest.config.addoption() should add the correct option to the config
- pytest.config.getopt_list() should correctly parse the added options
- pytest.config.dictConfig() should correctly convert the parsed options to a 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/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_plugin_integration.py::TestPluginHooksWithPytester::test_both_json_and_html_outputs` 87ms 8

AI ASSESSMENT

Scenario: Test generates both JSON and HTML reports for a plugin.

Why Needed: This test prevents regression in case the plugin does not generate either report type.

Key Assertions:

- The 'report.json' file exists at the specified path.
- The 'report.html' file exists at the specified path.
- The 'report.json' file is generated correctly by the plugin.
- The 'report.html' file is generated correctly by the plugin and matches the content of the 'report.json' file.

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/models.py	75 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	46 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-

	177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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	117 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226-227, 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, 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_integration.py::TestPluginHooksWithPytester::test_collection_finish_counts_items` 59ms 7

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

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/models.py	75 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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_plugin_integration.py::TestPluginHooksWithPytester::test_creates_nested_directory` 58ms 7

AI ASSESSMENT

Scenario: Test that output directories are created if missing.

Why Needed: Prevents a bug where the test fails due to an empty directory being reported as created.

Key Assertions:

- The `report.json` file should be present in the 'nested' directory.
- The `report.json` file should not be empty.
- The `report.json` file should exist in the specified path.
- The test should fail if an empty directory is reported as created.
- The test should pass if a non-empty directory is reported as created.

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/models.py	81 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 235-237, 239, 241, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516-518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160,

	164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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, 222-223, 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_plugin_integration.py::TestPluginHooksWithPytester::test_fixture_error_captured` 169ms 7

AI ASSESSMENT

Scenario: Test that fixture errors are captured in report.

Why Needed: Prevents a regression where the test fails with an unhandled error.

Key Assertions:

- The 'summary' key in the report contains the number of failed fixtures.
- The 'error' key in the 'summary' dictionary is set to 1 if there are any errors.
- The 'status' key in the 'summary' dictionary is set to 'error' if there are any errors.
- The 'name' key in the 'summary' dictionary contains a string indicating that an error occurred.
- The 'message' key in the 'summary' dictionary contains a message explaining why the test failed.
- The 'test_name' key in the 'summary' dictionary contains the name of the test that failed.

COVERAGE

src/pytest_llm_report/collector.py	50 lines (ranges: 78-79, 90, 93-94, 96, 99-103, 109-112, 114-115, 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	76 lines (ranges: 162, 166-171, 173-175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
src/pytest_llm_report/report_writer.py	110 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, 325, 327-328, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_plugin_integration.py::TestPluginHooksWithPytester::test_makereport_captures_all_outcomes` 172ms 7

AI ASSESSMENT

Scenario: Test pytest_runtest_makereport captures all outcomes.

Why Needed: This test prevents a potential regression where the report does not capture all outcomes due to an issue with the --llm-report-json flag.

Key Assertions:

- The 'passed' outcome should be included in the report.
- The 'failed' outcome should be included in the report.
- The 'skipped' outcome should be included in the report.
- All outcomes (passed, failed, skipped) should be captured by pytest_runtest_makereport.
- The report path should contain a file named 'report.json'.
- The report path should not be empty. If it is, pytester.runpytest will raise an error.

COVERAGE

src/pytest_llm_report/collector.py	55 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 106-107, 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, 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	76 lines (ranges: 162, 166-171, 173-175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
src/pytest_llm_report/report_writer.py	109 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, 317-322, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_plugin_integration.py::TestPluginHooksWithPytester::test_no_report_when_disabled` 55ms 3

AI ASSESSMENT

LLM 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	44 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	114 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225-226, 363-364, 367-368, 371-373, 384, 388, 407-408, 424-425)

PASSED `tests/test_plugin_integration.py::TestPluginHooksWithPytester::test_pdf_option_enables_plugin` 598ms 8

AI ASSESSMENT

Scenario: Test that --llm-pdf option enables the plugin.

Why Needed: Prevents regression in plugin key validation and ensures successful execution of plugin logic when only --llm-pdf is provided.

Key Assertions:

- The test verifies that the `test_pass` function returns True without any errors.
- The test verifies that the JSON file generated by the plugin is present if we also ask for it (proving the plugin key validation passed).
- The test checks for a warning about Playwright if applicable, or ensures execution doesn't error out on 'disabled plugin'.

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/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426, 428-430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388,

	407, 411-413, 424, 428, 431, 433-434)
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	105 lines (ranges: 67-73, 85-86, 98-100, 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, 414-416, 424-429, 432, 434-435, 448, 453, 455, 458-462, 470-471)

PASSED `tests/test_plugin_integration.py::TestPluginHooksWithPytester::test_session_start_records_time` 61ms 7

AI ASSESSMENT

Scenario: Verify that the pytest_sessionstart records start time is correctly reported in the report.json file.

Why Needed: This test prevents a potential bug where the start time of the session is not accurately recorded in the report.json file.

Key Assertions:

- The 'start_time' key should be present in the run_meta dictionary with the correct value.
- The 'start_time' value should match the actual start time of the session.
- The 'start_time' value should be a valid datetime object.
- The report.json file should contain the 'start_time' key and its corresponding value.
- The 'start_time' value should not be None or empty.
- The 'start_time' value should be in the correct format (e.g., YYYY-MM-DD HH:MM:SS).
- The pytest_sessionstart records should start recording time as soon as the test is run.

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/models.py	75 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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_plugin_integration.py::TestPluginIntegration::test_llm_context_marker 1ms 2

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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: 363-364, 367, 371-373)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the requirement marker could be misinterpreted as an error.

Key Assertions:

- The `requirement_marker` function should not raise any exceptions when called with valid input.
- The `requirement_marker` function should not throw any errors when executed without arguments.
- The `requirement_marker` function should return a boolean value indicating whether the requirement is met or not.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

PASSED `tests/test_plugin_integration.py::TestReportGeneration::test_report_writer_integration` 36ms 6

AI ASSESSMENT

Scenario: The test verifies that the report writer integrates with pytest_llm_report models correctly.

Why Needed: This test prevents a regression where the report writer may not generate reports for tests that fail or have errors.

Key Assertions:

- Verify that the report writer writes a JSON file containing the correct summary statistics.
- Verify that the report writer includes all nodes in the report.
- Verify that the report writer includes the expected error message in case of an error.
- Check if the total number of tests is 2 (passed + failed) as expected.
- Check if only test_a.py and test_b.py are included in the HTML report as expected.
- Verify that the report writer can write a valid JSON file even when there are no tests to generate reports for.

COVERAGE

src/pytest_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	81 lines (ranges: 162, 166-171, 173-175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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)

📄 tests/test_plugin_maximal.py

26 tests

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_disabled` 2ms 2

AI ASSESSMENT

Scenario: Test that collectreport skips when disabled and pytest_collectreport is mocked correctly.

Why Needed: This test prevents a regression where collectreport fails to run due to the plugin being disabled.

Key Assertions:

- The `pytest_collectreport` function is called with the `_enabled_key` as an argument and `False` as its second argument.
- The `get` method of `stash.get` is called on the mock report object with the `_enabled_key` key.
- The `session.config.stash.get` method is called with the `_enabled_key` key and `False` as its second argument.
- The `assert_called_with` method is called on the mock report object to verify it was called correctly.
- The `pytest_collectreport` function does not raise an exception when called with a disabled collectreport.
- The `pytest_collectreport` function does not modify the pytest session configuration.
- The `pytest_collectreport` function does not raise an exception when called without a mock report object.
- The `_enabled_key` is correctly set to `'collectreport'` on the mock report object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	10 lines (ranges: 363-364, 367, 371-373, 384-385, 391-392)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_enabled` 2ms 2

AI ASSESSMENT

Scenario: Verify that collectreport calls collector when enable is True.

Why Needed: Prevents a potential bug where the plugin does not collect reports even when enabled.

Key Assertions:

- The `pytest_collectreport` function should call `_collector_key` stash to get the collector instance.
- The `pytest_collectreport` function should call `_enabled_key` stash to get the enable flag.
- The `handle_collection_report` method of the collector instance should be called with a valid report object.
- The `session.config.stash.get` mock returns True for `_enabled_key` and False for `_collector_key` when enabled.
- The `handle_collection_report` method of the collector instance should not raise an exception when called with a valid report object.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	12 lines (ranges: 363-364, 367, 371-373, 384-385, 391, 395-397)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_no_session` 1ms 2

AI ASSESSMENT

Scenario: Verify that `pytest_collectreport` does not throw an exception when a mock report object is created without a 'session' attribute.

Why Needed: Prevent regression in the `pytest_collectreport` plugin, which may cause it to skip collect reports when no session is available.

Key Assertions:

- The function `pytest_collectreport(mock_report)` should not be called with an argument that has a 'session' attribute.
- The mock report object created without a 'session' attribute should not have a 'session' attribute.
- The mock report object created without a 'session' attribute should not raise any exceptions.
- The function `pytest_collectreport(mock_report)` should return without raising an exception when called with the mock report object.
- The function `pytest_collectreport(mock_report)` should call the original `collect` method on the mock report object if it exists.
- The function `pytest_collectreport(mock_report)` should not modify the mock report object in any way.

COVERAGE

`src/pytest_llm_report/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: 363-364, 367, 371-373, 384, 388)`

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_session_none` 1ms 2

AI ASSESSMENT

Scenario: Verify that the `pytest_collectreport` function does not throw an exception when a `None` session is passed.

Why Needed: Prevent regression in case of a `None` session, ensuring consistent behavior across different test environments.

Key Assertions:

- The `pytest_collectreport` function should not raise an error when given a `None` session.
- The `session` attribute of the mock report object should be set to `None` without any exceptions.
- No other errors or warnings should be raised in the test environment.
- The `pytest_collectreport` function should behave as expected when given a `None` session, without any side effects.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	8 lines (ranges: 363-364, 367, 371-373, 384, 388)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_llm_enabled_warning` 3ms 3

AI ASSESSMENT

Scenario: Verify that the 'pytest_llm_report' plugin raises a warning when LLM is enabled in the pyproject.toml file.

Why Needed: This test prevents a potential bug where the 'pytest_llm_report' plugin does not raise an error when LLM is enabled, potentially leading to silent failures or unexpected behavior.

Key Assertions:

- The 'pytest_llm_report' plugin raises a warning in the pyproject.toml file when LLM is enabled.
- The warning message is related to the 'LLM' setting in the pyproject.toml file.
- The warning is not suppressed by any configuration options provided by the 'pytest_llm_report' plugin.
- The error is only raised for the specified test module and function.
- The warning does not affect the overall functionality of the test suite.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	86 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-293, 295-296, 300-304, 308, 310, 312, 316, 320, 324, 328-330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	30 lines (ranges: 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184-186, 188-189, 193-195, 363-364, 367, 371-373)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_validation_errors` 3ms 3

AI ASSESSMENT

Scenario: Testing the `pytest_configure` function to ensure it raises a `UsageError` when encountering invalid configuration files.

Why Needed: This test prevents potential regression in the plugin's behavior when encountering malformed configuration files.

Key Assertions:

- The `pytest_configure` function should raise a `UsageError` with an appropriate error message when given an invalid pyproject.toml file.
- The error message should be informative and clearly indicate that the provided configuration is invalid.
- The test should fail when attempting to run pytest with an invalid configuration file, indicating a bug in the plugin's validation logic.
- The `UsageError` should be raised with a specific exception name (e.g., `UsageError`) rather than just raising any exception.
- The error message should include details about the invalid configuration file, such as its contents or location.
- The test should only fail when attempting to run pytest with an invalid configuration file and not when running it with valid 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	85 lines (ranges: 123, 163, 191, 194-197, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 276, 279-280, 288-293, 295-296, 300-302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	25 lines (ranges: 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-180, 363-364, 367, 371-373)

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 regression where the plugin might skip configuration on xdist workers due to an unhandled marker call.

Key Assertions:

- mock_config.addinivalue_line was not called before worker check
- addinivalue_line is still called for markers after 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: 150-152, 154-156, 158-160, 164-165, 363-364, 367, 371-373)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigureFallback::test_pytest_configure_fallback_load` 3ms 2

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	30 lines (ranges: 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184-186, 188-189, 193-195, 363-364, 367, 371-373)

PASSED `tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_cli_overrides_pyproject` 2ms 3

AI ASSESSMENT

Scenario: The test verifies that CLI options can override PyProject.toml settings.

Why Needed: This test prevents a potential bug where CLI options do not override PyProject.toml settings, leading to inconsistent configuration.

Key Assertions:

- pyproject.toml is created with the specified content.
- CLI options are correctly overridden by PyProject.toml settings.
- PyProject.toml settings are not overwritten by default.
- The test passes even when CLI options override PyProject.toml settings.
- The test fails when CLI options do not override PyProject.toml settings.
- The test ensures that CLI options take precedence over PyProject.toml settings.
- CLI options can be used to override specific settings in PyProject.toml.
- PyProject.toml is updated correctly after CLI option overrides.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	78 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-302, 304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414-415, 417-418, 420-421, 424-436, 440-448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_from_pyproject` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the plugin can successfully load all options from the provided pyproject.toml file.

Why Needed: This test prevents a potential issue where plugins may not be able to access or parse configuration files in certain scenarios.

Key Assertions:

- pyproject.toml should be created with the correct content.
- The plugin should be able to read and parse the pyproject.toml file correctly.
- All options specified in the pyproject.toml file should be accessible and usable by the plugin.
- Any configuration settings or values defined in the pyproject.toml file should be applied to the plugin's configuration.
- The plugin's configuration should not be corrupted or lost due to issues with the pyproject.toml file.
- Error messages from the plugin should indicate a successful load of the config from pyproject.toml.
- The plugin's behavior and functionality should remain unaffected by issues with the pyproject.toml file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	68 lines (ranges: 123, 163, 276, 279-280, 288-293, 295-296, 300-304, 308, 310, 312, 316, 320, 324, 328-330, 332, 334, 336, 340, 342, 346, 348, 350-352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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 regression where the plugin's terminal summary is not properly handled when it is disabled.

Key Assertions:

- The stash.get method of the stash object was called with _enabled_key and False.
- The stash.get method of the stash object was called with _enabled_key and True.
- The stash.get method of the stash object was not called at all when _enabled_key is False.
- The stash.get method of the stash object was called once with _enabled_key as None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	9 lines (ranges: 221, 225-226, 363-364, 367, 371-373)

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 correctly.

Why Needed: This test prevents a regression where the plugin does not skip terminal summaries on xdist workers due to incorrect configuration.

Key Assertions:

- The `pytest_terminal_summary` function returns None for mock_config.workerinput = {'workerid': 'gw0'}
- The `terminal_summary_worker_skip` method is called with a mock result of None
- The plugin does not skip terminal summaries on xdist workers due to incorrect configuration

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 221-222, 363-364, 367, 371-373)

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::testload_config` 2ms 3

AI ASSESSMENT

LLM 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`	`30 lines (ranges: 123, 163, 276, 279-280, 288-290, 414-415, 417-418, 420-421, 424-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_disabled` 2ms 2

AI ASSESSMENT

Scenario: Test makereport skips when disabled.

Why Needed: The test prevents a regression where the plugin does not report any errors when makereport is disabled.

Key Assertions:

- mock_item.config.stash.get() returns False
- mock_call.call() should have been called with a mock result
- mock_outcome.get_result().should.have.been.called_with(mocked_result)
- gen.send(mock_outcome).should.have.been.called_once_with
- mock_call.call().should.have.been.called_twice
- mock_item.config.stash.get() should have been called twice
- mock_outcome.get_result().should.have.been.called_twice

COVERAGE

`src/pytest_llm_report/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: 363-364, 367-368, 371-373)`

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_enabled` 2ms 2

AI ASSESSMENT

Scenario: Test that makereport calls collector when enabled.

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

Key Assertions:

- The `pytest_runtest_makereport` function should be able to find the `_collector_key` key in the mock item's configuration.
- The `pytest_runtest_makereport` function should be able to find the `_enabled_key` key in the mock item's configuration and return True if it is enabled.
- The `pytest_runtest_makereport` function should call the `handle_runtest_logreport` method on the mock collector with the provided report and item.
- The `handle_runtest_logreport` method of the mock collector should be called once with the provided report and item.
- The `get_result` method of the mock outcome should return a result that is not None when the test is run.
- The `send` method of the mock outcome should not raise an exception when the test is run successfully.
- The `stash_get` function of the mock item's configuration should be able to find the `_enabled_key` and `_collector_key` keys.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_disabled` 1ms 2

AI ASSESSMENT

Scenario: Test that collection_finish is skipped when disabled.

Why Needed: The test prevents a potential regression where collection_finish may not be executed correctly if it's disabled.

Key Assertions:

- mock_session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_collection_finish(mock_session) was called with the correct arguments
- collection_finish is skipped when stash.get 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/plugin.py	8 lines (ranges: 363-364, 367, 371-373, 407-408)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_enabled` 2ms 2

AI ASSESSMENT

Scenario: Test that collection_finish is called when enabled.

Why Needed: This test prevents a potential regression where the collector is not called when collection_finish is enabled.

Key Assertions:

- The stash_get function returns True for _enabled_key and mock_collector.
- The stash_get function returns mock_collector for _collector_key.
- collection_finish is called once with mock_session.items.
- mock_collector.handle_collection_finish is called once with mock_session.items.
- mock_collector.handle_collection_finish is not called twice when collection_finish 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	10 lines (ranges: 363-364, 367, 371-373, 407, 411-413)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_disabled` 1ms 2

AI ASSESSMENT

Scenario: Test that pytest_sessionstart skips when disabled and checks enabled status.

Why Needed: To prevent a regression where pytest_sessionstart fails to check the plugin's enabled status when it is disabled.

Key Assertions:

- mocked session config stash.get was called with _enabled_key and False
- mocked session config stash.get was called with _enabled_key and 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	8 lines (ranges: 363-364, 367, 371-373, 424-425)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_enabled` 1ms 3

AI ASSESSMENT

Scenario: Test that sessionstart initializes collector when enabled and creates a stash with the necessary keys.

Why Needed: This test prevents regression where pytest_sessionstart is not initialized or does not create a stash with the required keys.

Key Assertions:

- The _collector_key should be present in the mock_stash dictionary.
- The _start_time_key should also be present in the mock_stash dictionary.
- If sessionstart is called without enabling pytest_sessionstart, the collector should not be created.
- If sessionstart is called with an invalid stash, it should raise an error.
- The stash dictionary should have the correct keys: _enabled_key and _config_key.
- The stash dictionary should have a valid Config object as its value for _config_key.
- The stash dictionary should not have any other keys besides _enabled_key and _config_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: 123, 163)
src/pytest_llm_report/plugin.py	11 lines (ranges: 363-364, 367, 371-373, 424, 428, 431, 433-434)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption` 3ms 2

AI ASSESSMENT

Scenario: Test pytest_addoption adds expected arguments to the command line.

Why Needed: This test prevents a bug where pytest_addoption does not add the 'LLM-enhanced test reports' and 'LLM-coverage-source' options to the command line if they are not provided.

Key Assertions:

- parser.getgroup.assert_called_with('llm-report', 'LLM-enhanced test reports')
- calls.any('--llm-report in args[0]')
- calls.any('--llm-coverage-source in args[0]')

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`84 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, 121-125, 127-131, 133-137, 363-364, 367, 371-373)`

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption_no_ini` 2ms 2

AI ASSESSMENT

Scenario: Test verifies that pytest_addoption no longer adds INI options.

Why Needed: This test prevents a regression where pytest_addoption would add INI options to the plugin.

Key Assertions:

- The `addini` method of the parser is not called.
- The `parser.addini` call is removed from the test.
- The `pytest_addoption.parser.addini` method is no longer available in this version of pytest_llm_report.plugin.
- The plugin does not add INI options to the terminal summary if `pytest_addoption.no_ini` is set.
- When `pytest_addoption.no_ini` is set, the plugin should not modify the terminal summary.
- The plugin's behavior changes when `pytest_addoption.no_ini` is set.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`84 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, 121-125, 127-131, 133-137, 363-364, 367, 371-373)`

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_coverage_calculation` 3ms 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	53 lines (ranges: 221, 225, 229, 232, 251-252, 254, 256, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-288, 290, 292-295, 307-308, 313-314, 341-351, 363-364, 367, 371-373)

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: This test prevents regression in case the 'llm' provider is not properly configured or if the report HTML file is corrupted.

Key Assertions:

- Verify that the correct configuration is passed to pytest_terminal_summary.
- Check if the mock stash matches the expected stash.
- Assert that the annotate_tests method is called once with the provided config.
- Verify that the correct model name is used for the LLM provider.
- Check if the get_provider method returns the correct provider instance.
- Verify that the mock writer class is created correctly and has a return value of mock_writer.
- Assert that the annotate method is called once with the provided config.
- Verify that the mock terminal reporter's stats dictionary contains the expected key-value pair.

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	59 lines (ranges: 221, 225, 229, 232, 251-252, 254, 256, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307-308, 313-316, 319, 321, 324-326, 333-338, 341-351, 363-364, 367, 371-373)

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:

- The stash `_enabled_key` is set to `True` before calling `pytest_terminal_summary()`
- The stash `_config_key` contains the configuration object `cfg` after calling `pytest_terminal_summary()`
- The coverage map returned by `mock_mapper.map_coverage()` does not contain any coverage data when called with a mock reporter and no collector present

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	45 lines (ranges: 221, 225, 229, 232, 251-252, 254, 256, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_with_aggregation` 2ms 3

AI ASSESSMENT

LLM 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: 123, 163)
src/pytest_llm_report/plugin.py	21 lines (ranges: 221, 225, 229, 232-233, 235-236, 239-240, 242, 244-248, 363-364, 367, 371-373)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummaryErrors::test_terminal_summary_coverage_error` 4ms 3

AI ASSESSMENT

Scenario: Test coverage calculation error when loading coverage map.

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

Key Assertions:

- The `pytest_terminal_summary` function is called with a mock `CoverageMapper` instance.
- A warning is raised for failed coverage computation.
- The `load` method of the `Coverage` class raises an exception.
- The `coverage_map` attribute of the `CoverageMapper` instance is not set to None.
- The `report_writer` attribute of the `ReportWriter` instance is not set to None.
- The `MagicMock()` object passed to `pytest_terminal_summary` has a `workerinput` attribute that is deleted.
- The `Config` object passed to `pytest_terminal_summary` has a `stash` attribute with a non-None value.
- The `CoverageMapper` instance has a `coverage_map` attribute set to an empty dictionary.

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: 123, 163, 252)
src/pytest_llm_report/plugin.py	52 lines (ranges: 221, 225, 229, 232, 251-252, 254, 256, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-288, 298-301, 307-308, 313-314, 341-351, 363-364, 367, 371-373)

📄 tests/test_prompts.py

6 tests

PASSED `tests/test_prompts.py::TestContextAssembler::test_assemble_balanced_context` 7ms 4

AI ASSESSMENT

Scenario: Test assembling a balanced context for `test_assemble_balanced_context` test function

Why Needed: Prevents regression due to unbalanced contexts, where the assembler may not correctly assemble the dependencies.

Key Assertions:

- The 'utils.py' file is present in the assembled context.
- The 'def util()' function is found in the 'utils.py' module of the assembled context.
- The coverage entry for 'utils.py' indicates that it was covered by the test.
- The assembler correctly assembles the dependencies, including 'test_a.py' and 'utils.py'.
- The 'def util()' function is present in the correct location within the 'utils.py' module.

COVERAGE

src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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: Verifies that the ContextAssembler can assemble a complete context for a test file with a single test function.

Why Needed: This test prevents regression when the llm_context_mode is set to 'complete' and the test file has only one test function.

Key Assertions:

- The source code of the test file should contain the specified test function.
- The ContextAssembler should be able to assemble a complete context for the test file.
- The assembled context should include the specified test function.
- The assembler should report an error if the test file has multiple test functions or other non-test code.
- The assembler should not report any errors when assembling a complete context with only one test function.
- The assembler should be able to assemble a complete context for a test file with a single test function even if it is in a different module than the test function.
- The assembler should be able to assemble a complete context for a test file with multiple test functions and other non-test code.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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: Assembles a minimal context for testing `test_1` function in `test_a.py`.

Why Needed: This test prevents a potential regression where the minimal context is not correctly assembled, potentially leading to incorrect test results or errors.

Key Assertions:

- The 'test_1' function should be found in the source code of `test_a.py`.
- The assembly result should contain only the `test_1` function definition.
- The context object should be an empty dictionary, indicating no dependencies are being assembled.

COVERAGE

src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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: Verifies that the ContextAssembler does not exceed the specified context limits when assembling a test file.

Why Needed: This test prevents potential memory leaks or unexpected behavior due to excessive context size exceeding the configured limit.

Key Assertions:

- The assembled context contains only 'f1.py' and does not contain any truncated content.
- The length of the assembled context is within the specified limit (40 bytes).
- No truncation message is present in the assembled context.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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's _get_test_source method with edge cases.

Why Needed: This test prevents a potential bug where the assembler does not correctly handle non-existent files or nested test names with parameters.

Key Assertions:

- The function returns an empty string when given a non-existent file path.
- The function correctly extracts the test name and parameter from the source code of a nested test.
- The function includes all necessary keywords in its output.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 LLM.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly excludes important files, leading to incorrect results or errors.

Key Assertions:

- assert assembler._should_exclude('test.pyc') is True
- assert assembler._should_exclude('secret/key.txt') is True
- assert assembler._should_exclude('public/readme.md') is False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	5 lines (ranges: 33, 191-194)

📄 tests/test_prompts_coverage.py

12 tests

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_assemble_minimal_mode` 1ms 4

AI ASSESSMENT

Scenario: Test assemble minimal mode returns no context files.

Why Needed: Prevents a regression where assemble in minimal mode does not generate any context files.

Key Assertions:

- context_files is an empty list when config llm_context_mode = 'minimal'.
- test_source contains the function definition of test_foo.
- test_source does not contain any import statements or other code that could be used to create a context file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_assemble_with_context_override` 1ms 4

AI ASSESSMENT

LLM 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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	50 lines (ranges: 33, 49, 52, 55, 58, 60-61, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-109, 111-112, 116, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-155, 158-159, 163, 191-192, 194)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_balanced_context_excludes_patterns` 1ms 4

AI ASSESSMENT

Scenario: Test balanced context excludes files matching exclude patterns.

Why Needed: This test prevents a regression where the LLM context is not excluded from coverage when it should be.

Key Assertions:

- The function `assembler._get_balanced_context` returns an empty dictionary when no matching files are found in the specified directory.
- The file path of the secret configuration file matches one of the exclude patterns.
- The number of lines in the secret configuration file is exactly 1.
- The line count of the secret configuration file is exactly 1.
- No matching files are found in the specified directory.
- The LLM context should be excluded from coverage when it contains a pattern that matches one of the exclude patterns.
- The output of `assembler._get_balanced_context` includes a message indicating that no matching files were found.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	16 lines (ranges: 33, 132, 135-138, 140-141, 144-145, 148-149, 163, 191-193)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_balanced_context_file_not_exists` 1ms 4

AI ASSESSMENT

Scenario: Test that a balanced context is skipped for a non-existent file.

Why Needed: Prevents a potential bug where the test passes when a file does not exist in the repository root.

Key Assertions:

- The `ContextAssembler` should return an empty context for a non-existent file.
- The file path of the non-existent file should match the expected coverage entry.
- The line ranges and line count of the coverage entry should be correct for the non-existent file.
- The test case result should indicate that the test passed with no errors or warnings.
- The context returned by the `ContextAssembler` should not contain any nodes or edges.

COVERAGE

src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 132, 135-138, 140-141, 144-146, 163)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_balanced_context_max_bytes_limit` 1ms 4

AI ASSESSMENT

Scenario: Test that balanced context respects max bytes limit.

Why Needed: Prevents a potential memory leak by ensuring the content of the context is truncated when it exceeds the maximum allowed bytes.

Key Assertions:

- The length of the content in the source file `large_module.py` should be less than or equal to 120 bytes after truncation.
- A message indicating that the content was truncated should be present in the source file `large_module.py`.
- If the content is not truncated, it means the maximum allowed bytes limit has been exceeded and the content will be kept in memory.
- The content of the source file `large_module.py` should have a length less than 120 bytes after truncation.
- A message indicating that the content was truncated should be present in the source file `large_module.py` if it is truncated.
- If the content is not truncated, it means the maximum allowed bytes limit has been exceeded and the content will be kept in memory.
- The context of the test should have a length less than or equal to 120 bytes after truncation.
- A message indicating that the context was truncated should be present in the context of the test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	22 lines (ranges: 33, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-156, 158-159, 163, 191-192, 194)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_balanced_context_no_coverage` 1ms 4

AI ASSESSMENT

Scenario: Test balanced context with no coverage returns empty dict.

Why Needed: Prevents a regression where the ContextAssembler does not return any context for uncovered nodes in a test.

Key Assertions:

- The ContextAssembler returns an empty dictionary when there are no uncovered nodes in the test.
- The ContextAssembler does not raise an exception or throw an error when there are no uncovered nodes in the test.
- The ContextAssembler correctly identifies and returns an empty dictionary for uncovered nodes.
- The ContextAssembler's output is consistent with the expected result for balanced contexts without coverage.
- The ContextAssembler's output is consistent with the expected result for unbalanced contexts with coverage.
- The ContextAssembler does not return any context when there are no uncovered nodes in the test, as per the test's requirements.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	3 lines (ranges: 33, 132-133)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_balanced_context_reaches_max_bytes_before_file` 1ms 4

AI ASSESSMENT

Scenario: Test that loop exits when max bytes is reached before processing file.

Why Needed: This test prevents a potential bug where the ContextAssembler exceeds the maximum allowed bytes in its context without encountering any errors.

Key Assertions:

- The length of the balanced_context variable should be either 0 or 1, indicating that it has been truncated to accommodate only one file.
- If the number of files processed is greater than 1, the balanced_context variable should have a length equal to 1.
- If the ContextAssembler does not encounter any errors, the coverage report for both files should be empty.
- If the ContextAssembler encounters an error, the coverage report for the file that caused the error should be non-empty and include all lines in the file.
- The coverage report for the second file should also be non-empty and include all lines in the file.
- The ContextAssembler should not exceed the maximum allowed bytes in its context without encountering any errors.
- If the llm_context_bytes configuration is set to a value greater than 5, the ContextAssembler should still only process one file before hitting the limit.
- If the llm_context_file_limit configuration is set to a value less than or equal to 10, the ContextAssembler should not exceed the maximum allowed bytes in its context without encountering any errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	23 lines (ranges: 33, 132, 135-138, 140-142, 144-145, 148, 151-152, 154-156, 158-159, 163, 191-192, 194)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_complete_context_delegates_to_balanced` 1ms 4

AI ASSESSMENT

Scenario: The test verifies that the `ContextAssembler` correctly delegates to a balanced configuration.

Why Needed: This test prevents a potential regression where incomplete contexts are used with unbalanced configurations.

Key Assertions:

- The function `_get_complete_context` should return a context object containing the specified file.
- The context object should contain the same logic as the `balanced` configuration.
- The coverage entry for `module.py` should indicate that it was executed in complete contexts.
- The test case should be able to pass without any errors when using an unbalanced configuration.
- The context should not be empty or None when passed to a function that requires a complete context.
- The function `_get_complete_context` should handle cases where the specified file is not found.
- The coverage entry for `module.py` should indicate that it was executed in complete contexts, even if only one line was executed.

COVERAGE

src/pytest_llm_report/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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	22 lines (ranges: 33, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-155, 158-159, 163, 180, 191-192, 194)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_get_test_source_empty_nodeid` 1ms 4

AI ASSESSMENT

Scenario: Test _get_test_source with empty nodeid returns empty string.

Why Needed: Prevents a potential bug where an empty nodeid in the config causes the test to fail.

Key Assertions:

- The input string is split into parts and any part after the split point is considered as empty.
- An empty string should be returned when no valid nodeids are found.
- The assembler's _get_test_source method should handle empty strings correctly.
- Without this test, an unexpected error might occur due to the potential bug.
- This test ensures that the assembler's behavior remains consistent with expected requirements.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	9 lines (ranges: 33, 78-79, 82-83, 86-89)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_get_test_source_extraction_stops_at_next_def` 1ms 4

AI ASSESSMENT

LLM 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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	25 lines (ranges: 33, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 114, 116)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_get_test_source_file_not_exists` 1ms 4

AI ASSESSMENT

Scenario: Verifies that the _get_test_source method returns an empty string when a non-existent test file is provided.

Why Needed: Prevents a potential bug where the assembler does not handle cases with missing or invalid test files.

Key Assertions:

- The result of calling _get_test_source with 'nonexistent.py::test_foo' as the file path and tmp_path should be an empty string.
- The method should raise a ValueError when called with a non-existent file.
- The assembler should handle cases where the test file is not found or does not exist.
- The assembler should provide a meaningful error message indicating the missing file.
- The assembler should not silently ignore the test file and return an empty string instead.
- The method should raise an exception when called with a non-existent file, rather than returning 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/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	6 lines (ranges: 33, 78-79, 82-84)

PASSED `tests/test_prompts_coverage.py::TestContextAssemblerEdgeCases::test_get_test_source_with_class` 1ms 4

AI ASSESSMENT

Scenario: The test verifies that the `_get_test_source` method correctly extracts a function from a test file with proper indentation.

Why Needed: This test prevents a potential bug where the extracted function is not properly indented, leading to incorrect test source code.

Key Assertions:

- The function should be indented according to Python's official style guide (PEP 8).
- The function name should match the expected class name.
- The function body should contain at least one line of code that matches the expected class method.
- The function should not have any trailing whitespace or unnecessary indentation.
- The function should be a valid Python function with no syntax errors.
- The extracted function should be able to be called without raising an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/prompts.py	25 lines (ranges: 33, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 114, 116)

📄 tests/test_ranges.py

13 tests

PASSED `tests/test_ranges.py::TestCompressRanges::test_consecutive_lines` 1ms 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`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:

- The range '2-3' is correctly identified as a single range.
- The range '1-3' is correctly identified as a single range.
- The range '2-3' and '1-3' are not considered duplicates because they have different start values.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`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 the function incorrectly returns a non-empty string for an empty input list.

Key Assertions:

- The function should return an empty string when given an empty list as input.
- The function should not raise any exceptions or errors when given an empty list as input.
- The function should correctly handle and return an empty string for an empty input list.
- The function should preserve the original order of elements in the input list.
- The function should ignore non-compressed ranges when returning a compressed result.
- The function should not produce incorrect results for lists with multiple consecutive empty ranges.
- The function should correctly handle and return an empty string for a single-element input list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 compressing mixed ranges in the test_mixed_ranges function.

Why Needed: This test prevents regression when mixing ranges with single values.

Key Assertions:

- The range '1-3' should be included in the output because it spans two numbers.
- The range '5, 10-12' should also be included because it spans multiple numbers.
- The range '15' should not be included as it only contains a single number.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/ranges.py	16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)

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

AI ASSESSMENT

Scenario: Test that non-consecutive lines are correctly compressed into a single comma-separated string.

Why Needed: This test prevents regression when the input list contains non-consecutive line numbers.

Key Assertions:

- The function should return a comma-separated string of the input range values.
- The first and last values in the range should be included in the output.
- Any duplicate values within the range should be ignored.
- Non-consecutive values should not be separated by commas.
- The resulting string should have the same length as the original list.
- The function should handle empty input lists correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 test verifies that a single-line input does not use the range notation.

Why Needed: This test prevents a potential bug where the function incorrectly handles single-line inputs.

Key Assertions:

- Input should be a list of numbers
- Output should be the same as the original number
- Range notation should not be used

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/ranges.py	10 lines (ranges: 29, 33, 35-37, 39, 50, 52, 65-66)

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

AI ASSESSMENT

Scenario: tests/test_ranges.py::TestCompressRanges::test_two_consecutive

Why Needed: This test prevents a regression where the function incorrectly handles consecutive lines without range notation.

Key Assertions:

- The input list should be compressed to '1-2' using range notation.
- The output of `compress_ranges([1, 2])` should match '1-2'.
- The function should raise an error for non-consecutive lines without range notation.
- The function should handle consecutive lines correctly and return the correct compressed string.
- The input list should contain only integers.
- The output of `compress_ranges(['a', 'b'])` should match '1-2'.
- The function should not raise an error for non-consecutive lines like ['a', 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: 363-364, 367, 371-373)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

PASSED `tests/test_ranges.py::TestCompressRanges::test_unsorted_input` 1ms 3

AI ASSESSMENT

Scenario: Test 'test_unsorted_input' verifies that the function handles unsorted input correctly.

Why Needed: This test prevents a potential bug where the function would incorrectly group ranges when input is not sorted.

Key Assertions:

- The function should return the correct compressed range for an unsorted input, e.g. '1-3, 5'.
- The function should handle cases where the input list contains duplicate values and still produce the expected output.
- The function should correctly group ranges even if they are not consecutive (e.g. [2, 4] is considered a single 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: 363-364, 367, 371-373)
src/pytest_llm_report/util/ranges.py	16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)

PASSED tests/test_ranges.py::TestExpandRanges::test_empty_string 1ms 3

AI ASSESSMENT

Scenario: The test verifies that an empty string produces an empty list when expanding ranges.

Why Needed: This test prevents a potential bug where an empty string is incorrectly expanded to include all elements, potentially leading to incorrect results or errors.

Key Assertions:

- assert expand_ranges([]) == []
- assert expand_ranges(['a', 'b']) == ['a', 'b']
- assert expand_ranges(['a', 'b', 'c']) == ['a', 'b', 'c']
- assert expand_ranges(['x', 'y', 'z']) == ['x', 'y', 'z']
- assert expand_ranges(['a', 'b', 'c', 'd']) == ['a', 'b', 'c', 'd']

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 that the `expand_ranges` function correctly handles mixed ranges and singles.

Why Needed: This test prevents regression in cases where a single value is part of a range (e.g., '1-3') or when multiple values are separated by ranges (e.g., '5, 10-12').

Key Assertions:

- The function correctly splits the input string into individual numbers and range parts.
- It handles both single values within ranges and entire ranges separately.
- It preserves the original order of numbers in cases where a single value is part of a range.
- It correctly handles ranges with negative or zero start values (e.g., '10-12')
- It ignores empty strings as input (e.g., '1, 5, ', which would be split into ['1', '', '5'])
- It preserves the order of numbers when a range is specified after a single value (e.g., '1-3, 5, 10')
- It handles cases where multiple ranges are provided in a single string (e.g., '1-3, 5-7, 9-11')

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 tested with a valid range.

Why Needed: This test prevents the function from expanding ranges incorrectly when they are not in the format 'start-end'.

Key Assertions:

- The function should return a list of numbers between start and end.
- The start value should be greater than or equal to 0.
- The end value should be less than the start value plus one.
- The range should not include the start value if it is equal to the end value.
- The function should handle ranges with a single number correctly.
- The function should raise an error for invalid input (e.g. 'a-b' or '-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/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/ranges.py	10 lines (ranges: 81, 84-91, 95)

PASSED tests/test_ranges.py::TestExpandRanges::test_roundtrip 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `compress_ranges` and `expand_ranges` functions produce the same output for a given input.

Why Needed: This test prevents regression in cases where the order of elements in the compressed or expanded ranges changes.

Key Assertions:

- The original list is not modified when calling `compress_ranges()` and then `expand_ranges()`.
- The first element of the compressed range is equal to the first element of the original list.
- The second element of the compressed range is equal to the second element of the original list.
- The third element of the compressed range is equal to the fourth element of the original list.
- The last element of the expanded range is equal to the fifth element of the original list.
- The first element of the expanded range is equal to the first element of the original list.
- The second element of the expanded range is equal to the second element of the original list.
- The third element of the expanded range is equal to the fourth element of the original list.
- The last element of the expanded range is equal to the fifth element of the original 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: 363-364, 367, 371-373)
src/pytest_llm_report/util/ranges.py	27 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67, 81, 84-91, 93, 95)

PASSED `tests/test_ranges.py::TestExpandRanges::test_single_number` 1ms 3

AI ASSESSMENT

Scenario: The 'expand_ranges' function is expected to handle a single input ('5') and return a list containing only that number.

Why Needed: This test prevents the function from expanding multiple numbers into separate lists, which could lead to incorrect results or unexpected behavior.

Key Assertions:

- assert expand_ranges('5') == [5]
- assert expand_ranges('-5') == [-5]
- assert expand_ranges('10') == [10]
- assert expand_ranges('0') == [0]
- assert expand_ranges('1.2') == [1.2]
- assert expand_ranges('abc') == []
- assert expand_ranges('12345') == [12345]

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/ranges.py	7 lines (ranges: 81, 84-87, 93, 95)

📄 tests/test_render.py

9 tests

PASSED `tests/test_render.py::TestFormatDuration::test_milliseconds` 1ms 3

AI ASSESSMENT

Scenario: Test that the `format_duration` function correctly formats durations for millisecond intervals less than 1 second.

Why Needed: This test prevents a regression where durations less than 1 second are not formatted as milliseconds.

Key Assertions:

- The duration is formatted as 'xms' (e.g. '500ms'),
- The duration is equal to the input value multiplied by 1000 (e.g. '1ms'),
- The duration is less than or equal to 1 second (1000ms),

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 correctly formats durations in seconds for values greater than or equal to 1 second.

Why Needed: This test prevents a potential bug where the function does not handle durations of exactly 1 second correctly.

Key Assertions:

- The function should return '1.23s' when given an argument of 1.23 seconds.
- The function should return '60.00s' when given an argument of 60 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/render.py	2 lines (ranges: 65-66)

PASSED tests/test_render.py::TestOutcomeToCssClass::test_all_outcomes 1ms 3

AI ASSESSMENT

Scenario: All outcomes should map to CSS classes.

Why Needed: Prevents regression in CSS class mapping for different outcome statuses.

Key Assertions:

- outcome-to-css-class mapping should be consistent across all outcome statuses.
- outcome-to-css-class mapping should handle special cases like 'xfailed' and 'xpassed'.
- outcome-to-css-class mapping should not map to invalid or unknown CSS classes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/render.py	8 lines (ranges: 79-85, 87)

PASSED tests/test_render.py::TestOutcomeToCssClass::test_unknown_outcome 1ms 3

AI ASSESSMENT

Scenario: tests/test_render.py::TestOutcomeToCssClass::test_unknown_outcome

Why Needed: This test prevents a potential regression where the 'outcome-unknown' class is not applied to unknown outcomes.

Key Assertions:

- assert outcome_to_css_class('unknown') == 'outcome-unknown'
- assert outcome_to_css_class('valid') != 'outcome-unknown'
- assert outcome_to_css_class('invalid') != 'outcome-unknown'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/render.py	8 lines (ranges: 79-85, 87)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_basic_report` 1ms 3

AI ASSESSMENT

Scenario: Tests the rendering of a basic report with fallback HTML.

Why Needed: This test prevents regression that may occur when the report is rendered without fallback HTML.

Key Assertions:

- The document type declaration '' should be present in the rendered HTML.
- The title 'Test Report' should be included in the rendered HTML.
- The node IDs 'test::passed' and 'test::failed' should be found in the rendered HTML.
- The string 'PASSED' should appear in the rendered HTML for the passed nodes.
- The string 'FAILED' should appear in the rendered HTML for the failed nodes.
- The plugin version '0.1.0' should be present in the rendered HTML.
- The repository version '1.2.3' should be present in the rendered HTML.
- The text '**Plugin:** v0.1.0' should appear in the rendered HTML.
- The text '**Repo:** v1.2.3' should appear 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: 363-364, 367, 371-373)`
`src/pytest_llm_report/render.py`	`52 lines (ranges: 65-67, 79-85, 87, 121-124, 126-127, 131-132, 141-143, 145-153, 158-160, 196, 229-236, 239-245, 248-249)`

PASSED tests/test_render.py::TestRenderFallbackHtml::test_renders_coverage 1ms 3

AI ASSESSMENT

Scenario: Test renders coverage for fallback HTML.

Why Needed: Prevents a potential regression where the test fails to report coverage information when rendering fallback HTML.

Key Assertions:

- The 'src/foo.py' file should be included in the rendered HTML.
- There should be exactly 5 lines in the rendered HTML.
- The line count of the rendered HTML should match the number of lines in the 'src/foo.py' file (5).
- The coverage information should include the file path 'src/foo.py'.
- The coverage information should indicate that there are no additional lines beyond what is expected (0-4).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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: Verify that the test includes LLM annotations in the rendered HTML report.

Why Needed: This test prevents a potential security vulnerability where an attacker could manipulate the authentication flow to bypass authorization checks.

Key Assertions:

- The 'Tests login flow' scenario is present in the rendered HTML report.
- The 'Prevents auth bypass' reason is mentioned in the report.
- The LlmAnnotation object contains the required information (scenario and why_needed).
- The key assertion 'Tests login flow' is included in the HTML content.
- The key assertion 'Prevents auth bypass' is present in the HTML content.
- The LlmAnnotation object has the correct structure (nodeid, outcome, scenario, and why_needed).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 HTML includes both "XFailed" and "XPassed" summary entries.

Why Needed: This test prevents a regression where the xfailed/xpassed summary is not displayed correctly.

Key Assertions:

- The report root contains the "XFailed" and "XPassed" strings in its HTML output.
- The report root does not contain any "XPassed" string in its HTML output.
- The report root contains only "XFailed" string in its HTML output.
- The report root contains only "XPassed" string in its HTML output.
- The report root contains "XFailed" and "XPassed" strings in different positions in the HTML output.
- The report root does not contain "XPassed" string in a specific position (e.g. at the end of the page).
- The report root contains "XPassed" string but it is not displayed correctly (e.g. it is hidden behind other text).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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)

📄 tests/test_report_writer.py

19 tests

PASSED tests/test_report_writer.py::TestComputeSha256::test_different_content 1ms 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

PASSED tests/test_report_writer.py::TestComputeSha256::test_empty_bytes 1ms 3

AI ASSESSMENT

Scenario: Test 'test_empty_bytes' verifies that an empty bytes input produces consistent hash and correct length.

Why Needed: This test prevents a potential bug where different inputs produce different hashes due to differences in byte order or encoding.

Key Assertions:

- The two computed hashes should be equal (i.e., the same SHA256 hash).
- The length of the first computed hash should be 64 bytes (as expected for a SHA256 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: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

PASSED tests/test_report_writer.py::TestReportWriter::test_build_run_meta 5ms 4

AI ASSESSMENT

Scenario: Test the ReportWriter class to ensure it builds run metadata correctly.

Why Needed: This test prevents regressions where the report writer does not include version information in the build run metadata.

Key Assertions:

- The duration of the test should be 60 seconds.
- The pytest version should have a value.
- The plugin version should match the current __version__.
- The python version should match the current __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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 that the `build_summary` method counts all outcome types correctly.

Why Needed: This test prevents a regression where the total count of outcomes is not accurate due to missing or incorrectly counted 'x' outcomes.

Key Assertions:

- The sum of all outcome types should be equal to 6 (1 passed, 1 failed, 1 skipped, 1 xfailed, 1 xpassed, 1 error).
- The count of passed outcomes should be 1.
- The count of failed outcomes should be 1.
- The count of skipped outcomes should be 1.
- The count of xfailed outcomes should be 1.
- The count of xpassed outcomes should be 1.
- The count of error outcomes should be 1.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 123, 163)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)
`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 `build_summary` method correctly counts outcomes in a report.

Why Needed: This test prevents regression where the number of passed, failed, and skipped tests is not accurately reflected in the report's summary.

Key Assertions:

- asserts that the total count of all tests is equal to 4 (total number of tests)
- asserts that the number of passed tests is equal to 2 (number of tests with outcome 'passed')
- asserts that the number of failed tests is equal to 1 (number of tests with outcome 'failed')
- asserts that the number of skipped tests is equal to 1 (number of tests with outcome '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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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: The `ReportWriter` class initializes correctly with a provided configuration.

Why Needed: Without this test, the `ReportWriter` might not initialize with the expected configuration, potentially leading to unexpected behavior or errors.

Key Assertions:

- writer.config is set to the provided Config instance.
- writer.warnings is an empty list.
- writer.artifacts is an empty list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	3 lines (ranges: 156-158)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_assembles_tests` 5ms 4

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_assembles_tests verifies that the report includes all tests and provides summary statistics.

Why Needed: This test prevents regression by ensuring that the report writer correctly identifies and writes each test, even if no output paths are specified.

Key Assertions:

- The length of the report.tests list is equal to 2 (the number of tests)
- The total value of the report.summary.total property is equal to 2 (the number of tests)
- Each test in the report.tests list has a nodeid that matches one of the test nodes in the input list
- Each test result has an outcome that matches either 'passed' or 'failed'
- The summary statistics are accurate and provide meaningful information about the tests

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	93 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_includes_coverage_percent` 6ms 4

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class writes a report with an included coverage percentage.

Why Needed: This test prevents regression where the coverage percentage is not included in the report.

Key Assertions:

- The total coverage percentage of the report should be equal to the provided `coverage_percent` value.
- The `summary` object of the report should contain a `coverage_total_percent` attribute with the same value as the provided `coverage_percent` value.
- The `report` object's `summary` attribute should have a `coverage_total_percent` property set to the provided `coverage_percent` value.
- The coverage percentage is included in the report summary.
- The total coverage percentage of the report matches the given coverage percentage.
- The `ReportWriter` class correctly includes the coverage percentage in its reports.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	93 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-199, 202-206, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_includes_source_coverage` 5ms 4

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_includes_source_coverage verifies that the test writes a report with source coverage summary.

Why Needed: This test prevents regression in case the source code changes and the report writer needs to include the source coverage summary.

Key Assertions:

- The number of source coverage entries should be exactly 1.
- The file path of the first source coverage entry should match 'src/foo.py'.
- All source coverage entries should have a missing count between 0 and 7 inclusive.
- At least one source coverage entry should have a covered count greater than or equal to 8.
- The percentage of covered statements should be exactly 87.5%
- The covered ranges should match '1-4, 6-7'.
- All source coverage entries should have at least one missed 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/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	92 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_merges_coverage` 5ms 4

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_merges_coverage verifies that the report writer merges coverage into tests.

Why Needed: This test prevents a regression where the report writer does not merge coverage correctly, potentially leading to incorrect test results or missed coverage metrics.

Key Assertions:

- The report should have at least one coverage entry for each test.
- Each coverage entry should be associated with the correct file path.
- The first coverage entry in the report should correspond to the first test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	94 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186-189, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_atomic_write_fallback` 7ms 5

AI ASSESSMENT

Scenario: Test that the report writer falls back to direct write if atomic write fails.

Why Needed: This test prevents a regression where the report writer fails to write reports even when it encounters an error during atomic writes.

Key Assertions:

- The file "report.json" exists in the temporary directory.
- Any warnings with code 'W203' are 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: 235-237, 239, 241, 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516-518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	125 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506-507, 509-512, 515-516)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_creates_directory_if_missing` 7ms 5

AI ASSESSMENT

Scenario: Test verifies that the `ReportWriter` creates an output directory if it doesn't exist.

Why Needed: This test prevents a potential issue where the report writer does not create the necessary output directory even when it is missing.

Key Assertions:

- The output directory should be created with the correct path.
- The `report.json` file should be written to the correct location within the output directory.
- The `ReportWriter` instance should have been able to write the report successfully without raising an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	86 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 235-237, 239, 241, 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516-518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	123 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-477, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_ensure_dir_failure` 1ms 4

AI ASSESSMENT

Scenario: Test that a directory creation failure is captured as a warning with code W201.

Why Needed: To prevent the test from passing when the directory creation fails due to permission issues.

Key Assertions:

- The function should raise an OSError exception with code W201 for the given path.
- The function should catch any exceptions raised by mkdir and propagate them up to the caller as warnings with code W201.
- Any warnings raised during the directory creation process should be caught and reported 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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 the `get_git_info` function to handle Git command failures gracefully.

Why Needed: This test prevents a potential bug where the `get_git_info` function fails to retrieve Git information due to a missing or unresponsive Git installation, causing the report writer to fail.

Key Assertions:

- The `sha` variable is set to `None` after calling `get_git_info()`
- The `dirty` variable is set to `None` after calling `get_git_info()`

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`src/pytest_llm_report/report_writer.py`	`9 lines (ranges: 67-73, 85-86)`

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_html_creates_file` 36ms 6

AI ASSESSMENT

Scenario: Test verifies that the report writer creates an HTML file.

Why Needed: This test prevents a regression where the report writer does not create an HTML file, potentially causing issues with reports containing such files.

Key Assertions:

- The 'report.html' file should exist in the temporary directory.
- The 'report.html' file should contain expected content as specified by the tests.
- All test nodes ('test1', 'test2') should be found in the 'report.html' file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	115 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_html_includes_xfail_summary` 37ms 6

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:

- Asserts that 'XFAILED' and 'XFailed' keywords are present in the HTML content.
- Asserts that 'XPASSED' and 'XPassed' keywords are present in the HTML content.
- Verifies that both xfail and passed outcomes are included in the summary.
- Checks for correct formatting of xfail and passed keywords.
- Ensures that xfailed and xpasstoken are not present in the HTML content.
- Verifies that all test nodes have their respective outcome tags (xfailed, xpassed, xfailed, or xpassed).
- Asserts that no other keywords are present in the HTML 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/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	118 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321, 323-326, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_json_creates_file` 6ms 5

AI ASSESSMENT

Scenario: Test verifies that a JSON file is created with the report.

Why Needed: This test prevents regression where the report writer does not create a JSON file.

Key Assertions:

- The 'report.json' file should exist in the temporary directory.
- At least one artifact should be tracked for the report.
- The number of artifacts should be greater than zero.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	80 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	117 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_pdf_creates_file` 39ms 6

AI ASSESSMENT

Scenario: Test verifies that a PDF file is created when Playwright is available.

Why Needed: This test prevents regression where the report writer does not create a PDF file even if Playwright is available.

Key Assertions:

- The `report.pdf` attribute of the test result should exist and contain a valid path to the created PDF file.
- Any artifacts created by the report writer should have paths that match the expected paths for the created PDF file.
- The `report.pdf` attribute of the test result should not be empty or None.
- The `report.pdf` attribute of the test result should contain a valid path to the created PDF file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	125 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230-231, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 401, 410, 412, 414-423, 434-435, 437-443, 448, 453, 455, 458-462, 470-471)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_pdf_missing_playwright_warns` 6ms 4

AI ASSESSMENT

Scenario: Test that a warning is raised when the Playwright module is missing for PDF output.

Why Needed: This test prevents a potential issue where the report writer does not warn users about missing Playwright for PDF output.

Key Assertions:

- The file `report.pdf` should exist after running the test.
- Any warnings raised by the report writer should be of type WarningCode.W204_PDF_PLAYWRIGHT_MISSING.value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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)

📄 tests/test_report_writer_coverage.py

10 tests

PASSED `tests/test_report_writer_coverage.py::TestGetGitInfo::test_git_info_from_nonexistent_path` 2ms 3

AI ASSESSMENT

Scenario: Test verifies that `get_git_info` returns None when the path does not exist.

Why Needed: Prevents a potential bug where `get_git_info` incorrectly returns a Git SHA or status when the provided path is non-existent.

Key Assertions:

- The function `get_git_info(tmp_path)` should return `None` for a nonexistent path.
- The function `get_git_info(tmp_path)` should not attempt to retrieve any information from the nonexistent path.
- The function `get_git_info(tmp_path)` should raise an exception or return an error when the provided path is non-existent.
- The function `get_git_info(tmp_path)` should not perform any Git operations on a non-existent path.
- The function `get_git_info(tmp_path)` should handle the case where the path does not exist without raising an exception.
- The function `get_git_info(tmp_path)` should return `None` for a nonexistent directory (not just file).
- The function `get_git_info(tmp_path)` should raise an exception when attempting to read from a non-existent Git repository.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	9 lines (ranges: 67-73, 85-86)

PASSED `tests/test_report_writer_coverage.py::TestGetGitInfo::test_git_info_from_valid_repo` 5ms 3

AI ASSESSMENT

Scenario: Test getting git info from a valid repository should not crash and return the expected information.

Why Needed: This test prevents a potential bug where the function crashes if the input is invalid (e.g., not a git repo).

Key Assertions:

- The function gets the commit hash or returns None if it's not in a valid git repository.
- The function does not crash when given an invalid input.
- The function returns the expected information for a valid git repository.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`src/pytest_llm_report/report_writer.py`	`16 lines (ranges: 67-74, 76-81, 83-84)`

PASSED `tests/test_report_writer_coverage.py::TestGetPluginGitInfo::test_plugin_git_info_fallback` 1ms 3

AI ASSESSMENT

Scenario: Test that a plugin's Git info fallback works as expected when the import fails.

Why Needed: This test prevents a regression where the fallback to the git runtime fails due to an import failure in the _git_info module.

Key Assertions:

- The function get_plugin_git_info() should return None or a string (sha) even if the import of _git_info fails.
- The function get_plugin_git_info() should still work via the git runtime fallback when the import of _git_info fails.
- The function assert that the returned SHA is either None or an instance of str, indicating a successful fallback.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	3 lines (ranges: 127-128, 130)

PASSED `tests/test_report_writer_coverage.py::TestGetPluginGitInfo::test_plugin_git_info_returns_values` 1ms 3

AI ASSESSMENT

Scenario: Test that plugin git info returns some values.

Why Needed: Prevents a potential crash by ensuring the test doesn't attempt to access an invalid Git object.

Key Assertions:

- The function get_plugin_git_info() should return either None or a string.
- The function get_plugin_git_info() should not attempt to access an invalid Git object (i.e., sha is None).
- The function get_plugin_git_info() should handle the case where runtime git info is available and return 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/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	3 lines (ranges: 127-128, 130)

PASSED `tests/test_report_writer_coverage.py::TestReportWriterAtomicWrite::test_atomic_write_fallback` 7ms 5

AI ASSESSMENT

Scenario: Test atomic write falls back to direct write on error.

Why Needed: This test prevents a regression where the report writer fails to write a report even when an atomic write operation fails.

Key Assertions:

- The `report.json` file should exist after the test.
- The `report.json` file should not be empty after the test.
- The `report.json` file should have the correct content (e.g. JSON structure, data types).
- The `report.json` file should not contain any errors or warnings.
- The `report.json` file should have a valid timestamp.
- The `report.json` file should be written to the specified path.
- The `report.json` file should not be deleted or moved during the test.
- The `report.json` file should have the correct permissions (e.g. read-only, write permissions).

COVERAGE

src/pytest_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	80 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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_coverage.py::TestReportWriterPDF::test_pdf_playwright_exception` 115ms 6

AI ASSESSMENT

Scenario: Test PDF generation when playwright raises exception (lines 424-432) and expected a warning about PDF failure.

Why Needed: Prevents regression where PDF generation fails due to playwright exception without raising an error.

Key Assertions:

- Mocked playwright context should have raised a RuntimeError on browser launch failure.
- Expected the writer to raise a Warning for PDF failure.
- The warning message should contain 'W201' indicating the code of the warning.
- The PDF generation process should not be successful due to the exception.
- The test should verify that the warning is present in the warnings list.
- The warning should have a specific code (in this case, 'W201').
- The warning message should contain the expected string ('Browser launch failed').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/render.py	26 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	28 lines (ranges: 156-158, 401, 410, 412, 414-416, 424-429, 432, 434-435, 448, 453, 455, 458-462, 470-471)

PASSED `tests/test_report_writer_coverage.py::TestReportWriterPDF::test_pdf_playwright_not_installed` 1ms 4

AI ASSESSMENT

Scenario: Test PDF generation when playwright is not installed.

Why Needed: Prevents a potential bug where the test fails due to missing playwright installation.

Key Assertions:

- The report writer should have been able to generate a PDF file despite playwright not being installed.
- A warning message indicating that playwright is missing should be displayed.
- No PDF file should have been created in the temporary 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: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	9 lines (ranges: 156-158, 401-405, 408)

PASSED `tests/test_report_writer_coverage.py::TestReportWriterPDF::test_resolve_html_source_creates_temp` 30ms 6

AI ASSESSMENT

Scenario: Test _resolve_pdf_html_source creates temp file when no HTML source is provided.

Why Needed: Prevents a potential bug where the report writer does not create a temporary file for non-existent HTML sources.

Key Assertions:

- The test verifies that the report writer creates a temporary file with the correct suffix (.html) even if no HTML path is configured.
- The test verifies that the created temp file exists and has the correct suffix.
- The test verifies that the created temp file does not contain any HTML content.
- The test checks for a clean-up of the original report file after creating the temporary file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/render.py	26 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	11 lines (ranges: 156-158, 448, 453, 455, 458-462)

PASSED `tests/test_report_writer_coverage.py::TestReportWriterPDF::test_resolve_html_source_missing_html_file` 30ms 6

AI ASSESSMENT

Scenario: Test _resolve_pdf_html_source when configured HTML doesn't exist.

Why Needed: Prevents a potential bug where the test fails due to an empty or missing HTML source file, causing the report writer to fall back to a temporary file instead of using the actual HTML source.

Key Assertions:

- The path returned by _resolve_pdf_html_source should be an absolute path to the nonexistent HTML file.
- The path returned by _resolve_pdf_html_source should exist after the test is run.
- The report writer should use the actual HTML source if it exists, instead of falling back to a temporary file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/render.py	26 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	13 lines (ranges: 156-158, 448-450, 453, 455, 458-462)

PASSED `tests/test_report_writer_coverage.py::TestReportWriterPDF::test_resolve_html_source_uses_existing` 1ms 4

AI ASSESSMENT

Scenario: Test _resolve_pdf_html_source uses existing HTML file.

Why Needed: Prevents a potential bug where the test fails if an existing HTML file is used as the source for the PDF report.

Key Assertions:

- The path of the resolved HTML file matches the expected existing HTML file.
- The `is_temp` flag is set to False, indicating that the report was generated successfully without creating a temporary file.
- The test verifies that an existing HTML file can be used as the source for the PDF report without causing 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/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/report_writer.py	7 lines (ranges: 156-158, 448-451)

📄 tests/test_report_writer_coverage_v2.py

2 tests

PASSED `tests/test_report_writer_coverage_v2.py::test_report_writer_ensure_dir_creation` 1ms 4

AI ASSESSMENT

Scenario: Ensure directory creation is performed when report HTML file exists.

Why Needed: The test prevents a potential issue where the report writer does not create the required directory if it already exists.

Key Assertions:

- tmp_dir.exists()
- any(w.code == 'W202') for w in writer.warnings
- tmp_dir / 'r.html' 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/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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` 10ms 5

AI ASSESSMENT

Scenario: The test verifies that the report_writer_metadata_skips function correctly skips metadata when reports are disabled.

Why Needed: This test prevents a regression where the report writer would include metadata even when reports are disabled.

Key Assertions:

- The 'start_time' key should be present in the metadata.
- The 'llm_model' key should not be present in the metadata if the report is disabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	36 lines (ranges: 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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)

📄 tests/test_schemas.py

2 tests

PASSED tests/test_schemas.py::TestAnnotationSchema::test_from_dict_full 1ms 3

AI ASSESSMENT

Scenario: Test from dictionary with all fields

Why Needed: Prevent regression in case of missing or malformed input data

Key Assertions:

- assert 200
- assert token

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_schemas.py::TestAnnotationSchema::test_to_dict_full 1ms 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	8 lines (ranges: 90-92, 94-98)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_smoke_pytester.py

15 tests

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_report_created` 89ms 8

AI ASSESSMENT

Scenario: The HTML report is generated correctly and contains the test function.

Why Needed: This test prevents a potential issue where the report may not contain the expected test function.

Key Assertions:

- The report path exists at `report.html`
- The content of the report includes `` and `test_simple`
- The report contains the test function `test_simple`

COVERAGE

src/pytest_llm_report/collector.py	39 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424-426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)

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` 126ms 8

AI ASSESSMENT

Scenario: test_html_summary_counts_all_statuses verifies that the HTML summary counts include all statuses.

Why Needed: This test prevents regression where the HTML summary counts are missing for certain statuses (e.g., XFailed, XPassed).

Key Assertions:

- The function `assert_summary(labels: list[str], expected: int)` checks if the count of each status in the labels matches the expected value.
- If a match is found with an unexpected label, it raises an AssertionError.
- If no match is found for all statuses, it raises an AssertionError with a message indicating that there are missing summary labels.
- The function `assert_summary(labels: list[str], expected: int)` also checks if the count of each status in the labels matches the expected value when using fallback patterns.
- If a match is found with an unexpected label, it returns without raising an AssertionError.
- If no match is found for all statuses, it raises an AssertionError with a message indicating that there are missing summary labels.
- The function `assert_summary(labels: list[str], expected: int)` checks if the count of each status in the labels matches the expected value when using fallback patterns and joins the labels into a single string.
- If no match is found for all statuses, it raises an AssertionError with a message indicating that there are missing summary labels.

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/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424-426, 428,

	430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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` 66ms 7

AI ASSESSMENT

Scenario: The JSON report is created and its existence and content are verified.

Why Needed: This test prevents a potential bug where the test fails to create a valid JSON report due to missing or malformed schema version information.

Key Assertions:

- The `schema_version` key in the report data should be present and contain the correct value (in this case, '1.0')
- The `summary` section of the report data should have exactly two items: 'total' with a count of 2 and 'passed' with a count of 1
- The `failed` item in the summary section should have a count of 1

COVERAGE

src/pytest_llm_report/collector.py	51 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-118, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 227-228, 230-236, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	76 lines (ranges: 162, 166-171, 173-175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160,

	164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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` 60ms 13

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report when a provider is enabled.

Why Needed: Prevents regressions by ensuring LLM annotations are present in the report.

Key Assertions:

- The test passes if the 'LLM annotations' key exists in the report with the correct value.
- The 'LLM annotations' key should contain a dictionary with the required keys ('scenario', 'why_needed', and 'key_assertions').
- The 'LLM annotations' key should be present in the report's JSON output.
- The 'LLM annotations' key should have the correct value for the 'why_needed' key, which is 'Prevents regressions'.
- The 'LLM annotations' key should contain a list of strings with the required keys ('asserts True').
- The 'LLM annotations' key should be present in the report's JSON output and have the correct value for each assertion.
- The 'key_assertions' key should contain a dictionary with the expected values for each assertion.

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	34 lines (ranges: 37-38, 41, 60, 62, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 164-168, 170-171, 175, 179-180, 183, 185-186, 188, 197, 205-206, 208)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/models.py	96 lines (ranges: 104-107, 109-111, 113, 115, 162, 166-171, 173, 175, 177, 179, 182, 184-186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413-425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	86 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-293, 295-296, 300-304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	172 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184-186, 188-189, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-316, 319, 321, 324-328, 331, 333-338, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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	110 lines (ranges: 55, 67-73, 85-86, 98-99, 102, 105-108, 113, 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` 88ms 13

AI ASSESSMENT

Scenario: The test verifies that an LLM error is reported in the generated HTML output.

Why Needed: This test prevents a regression where LLM errors are not surfaced in the generated HTML output.

Key Assertions:

- The function `test_pass()` should raise an assertion error when executed.
- The function `test_llm_error_is_reported` should generate an HTML report with an error message.
- The test should fail if the LLM errors are not surfaced in the generated HTML output.
- The test should verify that the error is reported at the top of the page.
- The error message should be a RuntimeError exception.
- The error message should include the string 'boom'.
- The test should also verify that the error message includes the function name `test_pass()`.

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	36 lines (ranges: 37-38, 41, 60, 62, 80-81, 87, 89, 92, 94-95, 98, 100-101, 106, 108, 110-111, 114, 129, 131, 164-

	168, 170-171, 175, 179-180, 183, 205-206, 208)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	86 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-293, 295-296, 300-304, 308, 310, 312, 316, 320, 324, 328, 330, 332, 334, 336, 340, 342, 346, 348, 350, 352, 354, 356, 358, 362, 366, 370, 372, 374, 380, 384, 390, 392, 398, 400, 414, 417, 420, 424-426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	172 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184-186, 188-189, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-316, 319, 321, 324-329, 333-338, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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	106 lines (ranges: 55, 67-73, 85-86, 98-99, 102, 105-108, 113, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-296, 298-299, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED tests/test_smoke_pytester.py::TestMarkers::test_llm_opt_out_marker 59ms 7

AI ASSESSMENT

Scenario: Test the LLM opt-out marker functionality.

Why Needed: Prevents regression in LLM opt-out marker recording and reporting.

Key Assertions:

- The test verifies that the LLM opt-out marker is recorded correctly.
- The test ensures that the LLM opt-out marker is not recorded for non-LLM tests.
- The test checks if the LLM opt-out marker is set to True for all tests.
- The test verifies that the LLM opt-out marker is not set to False for non-LLM tests.
- The test ensures that the LLM opt-out marker is recorded correctly even when run with --llm-report-json flag
- The test checks if the LLM opt-out marker is correctly recorded in the report.json file

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	76 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186-188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160,

	164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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 57ms 7

AI ASSESSMENT

Scenario: Test the requirement marker to ensure it is recorded correctly.

Why Needed: The test prevents a potential bug where the requirement marker is not properly recorded, potentially leading to incorrect test results or missed tests.

Key Assertions:

- The `pytest.mark.requirement` decorator should be applied to the test function with the required requirements.
- The `requirement` parameter of the `@pytest.mark.requirement` decorator should match the actual requirement name (REQ-001 and REQ-002 in this case).
- The `requirements` attribute of the test result data should contain the required requirements.
- The `reqs` list within the test result data should contain the required requirements as strings.
- The `REQ-001` and `REQ-002` strings should be present in the `reqs` list.

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	76 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194-196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
src/pytest_llm_report/report_writer.py	105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestOutcomes::test_multiple_xfail_outcomes` 66ms 7

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

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	76 lines (ranges: 162, 166-171, 173-175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)

src/pytest_llm_report/report_writer.py	108 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321, 323-324, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED tests/test_smoke_pytester.py::TestOutcomes::test_skip_outcome 58ms 7

AI ASSESSMENT

Scenario: Test that skipped tests are recorded and reported correctly.

Why Needed: This test prevents a regression where the 'skip' marker is not properly recorded or reported in the report.

Key Assertions:

- The number of skipped tests should be correctly counted in the report
- The skipped tests should be included in the report summary
- The skipped tests should have a correct status (e.g., 'skipped', 'not_skipped') in the report

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	76 lines (ranges: 162, 166-171, 173-175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260,

	263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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 61ms 7

AI ASSESSMENT

Scenario: Verifies that X failed tests are recorded in the report.

Why Needed: This test prevents regression where X failed tests are not properly reported.

Key Assertions:

- The 'summary' key in the report.json file should contain a count of X failed tests.
- The value of the 'xfailed' key in the report.json file should be equal to 1.
- The test function 'test_xfail()' should fail and produce an assertion error.

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	76 lines (ranges: 162, 166-171, 173-175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274,

	276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
src/pytest_llm_report/report_writer.py	108 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321, 323-324, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED tests/test_smoke_pytester.py::TestParametrization::test_parametrized_tests 62ms 7

AI ASSESSMENT

Scenario: The test verifies that parameterized tests are recorded separately and their results are reported in a JSON file.

Why Needed: This test prevents regression by ensuring that the pytester is able to record and report on all parameterized tests correctly.

Key Assertions:

- The total number of successful parameterized tests is 3 (1, 2, and 3).
- All parameterized tests are recorded in a JSON file named 'report.json'.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163-164, 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	76 lines (ranges: 162, 166-171, 173, 175-177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426-428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260,

	263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
src/pytest_llm_report/report_writer.py	105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_help_contains_examples` 51ms 3

AI ASSESSMENT

Scenario: The CLI help text should include usage examples.

Why Needed: This test prevents a potential bug where the help text is not displayed with examples.

Key Assertions:

- The `stdout.fnmatch_lines` method is used to match lines in the output.
- The `result.stdout` attribute is accessed to get the output of the `pytester.runpytest` function.
- The `--help` argument is passed to `pytester.runpytest` to run the test with help mode.
- The `stdout.fnmatch_lines` method is used to match lines that contain 'Example:*--llm-report*'.
- The `result.stdout` attribute is accessed to get the output of the `pytester.runpytest` function.
- The `--help` argument is passed to `pytester.runpytest` to run the test with help mode.
- The `stdout.fnmatch_lines` method is used to match lines that contain 'Example:*--llm-report*'.
- The `result.stdout` attribute is accessed to get the output of the `pytester.runpytest` function.
- The `--help` argument is passed to `pytester.runpytest` to run the test with help 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	44 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	104 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 363-364, 367, 371-373)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_markers_registered` 45ms 3

AI ASSESSMENT

Scenario: Test that LLM markers are registered and correctly matched in the output.

Why Needed: This test prevents a potential bug where the LLM marker is not detected or matched correctly, potentially leading to incorrect results or warnings.

Key Assertions:

- The `--markers` option should match the expected list of markers (`*llm_opt_out*, *llm_context*, and *requirement*`).
- The output should contain at least one line that matches each marker (e.g., `*llm_opt_out*`, `*llm_context*`, or `*requirement*`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	44 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	104 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 363-364, 367, 371-373)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_plugin_registered` 51ms 3

AI ASSESSMENT

Scenario: Verify that the plugin is registered correctly by checking if it can be run with --help flag.

Why Needed: This test prevents a potential bug where the plugin might not register properly or may throw an error when trying to use it with pytest11.

Key Assertions:

- The `pytester.runpytest` method returns True if the plugin is registered correctly and False otherwise.
- The output of `result.stdout.fnmatch_lines` contains at least one line that starts with '--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	44 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424, 426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	104 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 363-364, 367, 371-373)

PASSED `tests/test_smoke_pytester.py::TestSpecialCharacters::test_special_chars_in_nodeid` 90ms 8

AI ASSESSMENT

Scenario: Test that special characters in nodeid are handled correctly by Pytester.

Why Needed: This test prevents a potential crash and ensures the HTML generated is valid.

Key Assertions:

- The report file should exist after running pytester.runpytest with --llm-report.
- The report file should contain '' in its contents.
- Special characters (like <, &, etc.) should not be treated as invalid nodeids by Pytester.

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	1 lines (ranges: 162)
src/pytest_llm_report/options.py	45 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-280, 288-290, 414, 417, 420, 424-426, 428, 430, 432, 434, 436, 440, 442, 444, 446, 448, 452, 454)
src/pytest_llm_report/plugin.py	152 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, 121-125, 127-131, 133-137, 150-152, 154-156, 158-160, 164, 168-169, 171, 173, 176-177, 184, 193-195, 221, 225, 229, 232, 251-252, 259-260, 263-264, 266-267, 270-274, 276, 279-280, 282, 285-286, 307, 313-314, 341-351, 363-364, 367, 371-373, 384, 388, 407, 411-413, 424, 428, 431, 433-434)
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)

📄 tests/test_time.py

15 tests

PASSED `tests/test_time.py::TestFormatDuration::test_boundary_one_minute` 1ms 3

AI ASSESSMENT

Scenario: Tests the 'format_duration' function to ensure it formats a duration of exactly one minute.

Why Needed: This test prevents regression where the function might incorrectly format a duration longer than one minute.

Key Assertions:

- The result should be '1m 0.0s' (one minute and zero seconds).
- The result should not contain any leading zeros for minutes.
- The result should only include the unit 'm' for minutes, with no additional characters.
- The function should handle durations up to one minute correctly without producing incorrect results.
- Any non-numeric values in the input duration should be ignored and not used as input.
- The test should pass even if the input duration is exactly 60 seconds (one full minute).
- The function should return an error or raise a ValueError for invalid input durations.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

PASSED `tests/test_time.py::TestFormatDuration::test_microseconds_format` 1ms 3

AI ASSESSMENT

Scenario: Verifies that the `format_duration` function correctly formats sub-millisecond durations as microseconds.

Why Needed: This test prevents a potential bug where the format string is not properly escaped for non-ASCII characters, potentially leading to incorrect formatting of microsecond durations.

Key Assertions:

- The result should contain 'µs' in its string representation.
- The function should correctly convert 0.0005 seconds to '500µs'.
- The formatted string should match the expected output.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 returns a string representation of sub-second durations as milliseconds.

Why Needed: Prevents incorrect formatting of millisecond-based durations, potentially leading to incorrect timing measurements in tests.

Key Assertions:

- The result should contain 'ms' as a prefix.
- The result should be equal to the expected value '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: 363-364, 367, 371-373)
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 where the function might incorrectly format minutes as seconds instead of displaying them separately.

Key Assertions:

- The result contains 'm' (minutes) in its string representation.
- The result equals '1m 30.5s' (correctly formatted duration with minutes and 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: 363-364, 367, 371-373)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

PASSED tests/test_time.py::TestFormatDuration::test_multiple_minutes 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a scenario of formatting multiple minutes.

Why Needed: This test prevents regression in the `format_duration` function when it is called with a duration that includes more than one minute.

Key Assertions:

- The output should be '3m 5.0s' for a duration of 185.0 seconds.
- The function should correctly handle durations with multiple minutes.
- The function should not throw an error when given invalid input (e.g., negative time).
- The function should preserve the original order of decimal places in the input time.
- The function should handle cases where the input is a float or int.
- The function should support durations with different units (e.g., seconds, minutes, 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: 363-364, 367, 371-373)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

PASSED tests/test_time.py::TestFormatDuration::test_one_second 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of exactly 1 second.

Why Needed: This test prevents regression in the case where the input duration is less than or equal to 0.01 seconds, which could cause incorrect formatting.

Key Assertions:

- The output of the `format_duration(1.0)` function should be '1.00s'.
- The unit of the output should be 's' (seconds).
- The decimal part of the output should be '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: 363-364, 367, 371-373)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED tests/test_time.py::TestFormatDuration::test_seconds_format 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function to ensure it correctly formats seconds under a minute.

Why Needed: This test prevents regression when seconds are formatted under a minute, as it ensures the output is consistent with the expected format.

Key Assertions:

- The result contains the string 's' (indicating seconds),
- The result equals the expected value '5.50s',

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED `tests/test_time.py::TestFormatDuration::test_small_milliseconds` 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 1 millisecond.

Why Needed: This test prevents regression in handling very small durations, where the default format might not be accurate.

Key Assertions:

- The output should be '1.0ms' for a duration of 1 millisecond.
- The function should handle durations less than 1 second correctly.
- The function should return the correct unit ('ms') for small durations.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`src/pytest_llm_report/util/time.py`	`3 lines (ranges: 39, 41-42)`

PASSED `tests/test_time.py::TestFormatDuration::test_very_small_microseconds` 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a very small duration (1 microsecond).

Why Needed: This test prevents a potential issue where the function returns incorrect results for extremely short durations.

Key Assertions:

- The output of `format_duration(0.000001)` should be '1µs'.
- The value returned by `format_duration(0.000001)` is equal to '1' followed by 'µs'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`src/pytest_llm_report/util/time.py`	`2 lines (ranges: 39-40)`

PASSED tests/test_time.py::TestIsoFormat::test_formats_datetime_with_utc 1ms 3

AI ASSESSMENT

Scenario: Tests the `iso_format` function with a datetime object representing UTC time.

Why Needed: Prevents a potential bug where the function does not correctly format datetime objects with UTC timezone.

Key Assertions:

- The output of `iso_format(dt)` should be in the ISO 8601 format `YYYY-MM-DDTHH:MM:SS+HH:MM:SSZ`.
- The time zone offset should be correctly represented as `+00:00`.
- The date and time components should be separated by a space, not a comma.
- The time component should have the correct format `HH:MM:SS`.
- The UTC timezone offset should be included in the output.
- The function should handle cases where the input datetime object is naive (i.e., does not have a timezone information).
- The function should correctly handle cases where the input datetime object has an invalid time zone.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

PASSED tests/test_time.py::TestIsoFormat::test_formats_naive_datetime 1ms 3

AI ASSESSMENT

Scenario: Verifies that naive datetime is correctly formatted without timezone.

Why Needed: Prevents a potential issue where naive datetime formats are incorrectly assumed to be in the local timezone.

Key Assertions:

- The result of `iso_format(dt)` should match '2024-06-20T14:00:00'.
- The time zone information is not present in the output.
- The date and time are correctly formatted without any timezone offset.
- The datetime object itself does not contain any timezone information.
- No timezone offset is applied to the datetime when formatting.
- The resulting string does not include a timezone indicator (e.g., UTC, GMT).
- The output does not contain any timezone-related metadata (e.g., timezone name, offset).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 may not correctly handle dates with microseconds.

Key Assertions:

- The result of calling `iso_format(dt)` should contain the string '123456'.
- The result of calling `iso_format(dt)` should be a valid ISO format string.
- The microseconds in the datetime object should be included in the output.
- The resulting string should not have any leading zeros.
- The resulting string should not contain any invalid characters.
- The resulting string should match the expected value when converted to a string using `isoformat()`.
- The resulting string should be able to be parsed correctly by other functions that expect an ISO format string.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
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 time within UTC's current time.

Why Needed: This test prevents a potential issue where `utc_now()` might return an incorrect or outdated time due to network latency or system clock discrepancies.

Key Assertions:

- The result of `utc_now()` should be within the range `[before, after)` (inclusive) when compared to `before` and `after` respectively.
- The value of `result` should not exceed `after` by more than a certain tolerance (e.g. 1 second).
- The value of `result` should not be less than `before` by more than a certain tolerance (e.g. 1 second).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)

PASSED `tests/test_time.py::TestUtcNow::test_returns_datetime` 1ms 3

AI ASSESSMENT

Scenario: The function `utc_now()` returns a datetime object when called.

Why Needed: This test prevents a potential bug where the function might return an incorrect or invalid datetime object.

Key Assertions:

- result is an instance of `datetime`
- result has a valid timezone.
- result 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/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`
`src/pytest_llm_report/util/time.py`	`1 lines (ranges: 15)`

📄 tests/test_token_refresh.py

12 tests

PASSED `tests/test_token_refresh.py::TestTokenRefresher::test_command_failure` 1ms 3

AI ASSESSMENT

Scenario: Test TokenRefresher raises error on command failure.

Why Needed: This test prevents a bug where the TokenRefresher does not raise an exception when the 'get-token' command fails, potentially leading to unexpected behavior or silent failures.

Key Assertions:

- The function `get_token()` in the `TokenRefresher` class should raise a `TokenRefreshError` with an error message indicating that authentication failed.
- The error message returned by `get_token()` should include the string 'exit 1' to indicate that the command failed.
- The error message returned by `get_token()` should contain the string 'Authentication failed' to identify the specific reason for the failure.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	20 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 101-104, 113, 115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh.py::TestTokenRefresher::test_empty_output` 1ms 3

AI ASSESSMENT

Scenario: Test that TokenRefresher raises an error when given empty output.

Why Needed: This test prevents a potential bug where the TokenRefresher class does not raise an error when it encounters no tokens to refresh.

Key Assertions:

- refresher.get_token() is called with no arguments.
- the stdout of refresher.get_token() is empty.
- the stderr of refresher.get_token() is empty.
- the returncode of refresher.get_token() is 0 (success), but the expected error message 'empty output' is not present in the output.
- the output format of refresher.get_token() is set to 'text', which does not contain any tokens, so there should be no 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/llm/token_refresh.py`	`20 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 101, 107-109, 113, 115)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 363-364, 367, 371-373)`

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_force_refresh 1ms 3

AI ASSESSMENT

Scenario: Test that forcing a refresh bypasses the cache and updates the token with a new value.

Why Needed: This test prevents a regression where the TokenRefresher does not update the token when forced to refresh.

Key Assertions:

- The function `get_token()` returns the expected token value for both initial and force refresh scenarios.
- The `call_count` variable is incremented correctly after each call to `get_token()` with `force=True`.
- The new token returned by `get_token()` has a unique identifier ('token-2' in this case).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	25 lines (ranges: 59-60, 63, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132, 153-154, 156)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh.py::TestTokenRefresher::test_get_token_json_custom_key` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `TokenRefresher` class correctly retrieves a custom JSON key for authentication.

Why Needed: This test prevents a potential bug where the custom JSON key is not properly retrieved during token refresh.

Key Assertions:

- The output of the `get-token` command should be 'custom-key-token' when the `json_key` parameter is set to 'access_token'.
- The `subprocess.run` function should return a CompletedProcess object with an stdout argument containing the expected JSON string.
- The `subprocess.run` function should return a CompletedProcess object with an stderr argument that is empty.
- The `TokenRefresher.get_token` method should call the `fake_run` function and return the result of the `subprocess.run` function.
- The `json.dumps` function should be called on the result of the `subprocess.run` function to generate the expected JSON string.
- The `token` variable should hold the value 'custom-key-token' after calling the `get_token` method.
- The test should pass without any errors or exceptions.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/token_refresh.py`	29 lines (ranges: 59-60, 63, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132-135, 139, 143-144, 148)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh.py::TestTokenRefresher::test_get_token_json_format` 1ms 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	29 lines (ranges: 59-60, 63, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132-135, 139, 143-144, 148)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh.py::TestTokenRefresher::test_get_token_text_format` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `TokenRefresher` extracts the correct token from text output.

Why Needed: This test prevents a potential bug where the token is not extracted correctly due to an incorrect or missing output format.

Key Assertions:

- token == 'my-secret-token'
- stdout contains 'INFO: Processing...' and 'my-secret-token'
- stderr does not contain any error messages
- the `get_token()` method returns the correct token
- the token is extracted from the text output 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/token_refresh.py	25 lines (ranges: 59-60, 63, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132, 153-154, 156)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_invalid_json 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the TokenRefresher raises a TokenRefreshError when given invalid JSON.

Why Needed: This test prevents the TokenRefresher from incorrectly handling or crashing on non-JSON input data, ensuring it remains stable and functional.

Key Assertions:

- The get_token() method of the TokenRefresher raises a TokenRefreshError with the message 'json' when given invalid JSON.
- The error message includes the string 'json', indicating that the issue is related to JSON format.
- When an invalid JSON is passed, the test asserts that the error message contains the word 'json'.
- The test verifies that the error message does not contain any other relevant information about the input data.
- The error message is case-insensitive, allowing for different input formats to be considered as valid or invalid JSON.
- The test checks that the error message includes the string 'json' in a manner consistent with standard error messages.
- When an invalid JSON is passed, the test verifies that the error message does not contain any other relevant information about the input data.
- The test ensures that the TokenRefresher correctly raises a TokenRefreshError when given invalid JSON.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	25 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 101, 107-108, 111, 113, 115, 132-134, 149-150)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_invalidate 1ms 3

AI ASSESSMENT

Scenario: Test TokenRefresher.invalidate() clears cache and verifies that it correctly invalidates the cache.

Why Needed: This test prevents a potential bug where the TokenRefresher does not invalidate the cache after calling get_token(), which could lead to stale token values being returned.

Key Assertions:

- The function call count should increase by 1 when run() is called.
- The result of subprocess.run() should contain a 'token-' prefix in the stdout and stderr.
- The output format should be set to 'text' for get_token() calls.
- The invalidation of the cache should not affect token values returned after calling get_token().
- The function call count should decrease by 1 when refresher.invalidate() is called.
- The result of subprocess.run() should contain a 'token-' prefix in the stdout and stderr after refresher.invalidate() 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/llm/token_refresh.py	28 lines (ranges: 59-60, 63, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132, 153-154, 156, 160-162)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_missing_json_key 1ms 3

AI ASSESSMENT

Scenario: Test that TokenRefresher raises error when JSON key is missing.

Why Needed: Prevents a potential bug where the token refresh process fails due to a missing required JSON key.

Key Assertions:

- The 'token' key should be present in the output of the get_token method.
- A message indicating that the 'token' key was not found should be returned by the get_token method.
- The error message should include the word 'not found'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	28 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 101, 107-108, 111, 113, 115, 132-135, 139-141, 149)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_thread_safety 52ms 3

AI ASSESSMENT

Scenario: Test TokenRefresher thread safety by starting multiple threads concurrently and verifying that they all retrieve the same token.

Why Needed: This test prevents a potential bug where multiple threads trying to refresh tokens at the same time could result in different tokens being retrieved, potentially leading to inconsistencies in the system.

Key Assertions:

- All threads should acquire the lock before getting the token and release it after getting the token.
- The first thread to acquire the lock should get the token with value 'token-1'.
- No other thread should get a different token than 'token-1'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	28 lines (ranges: 59-60, 63-66, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132, 153-154, 156)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_timeout_handling 1ms 3

AI ASSESSMENT

Scenario: The test verifies that TokenRefresher handles command timeouts correctly.

Why Needed: This test prevents the potential bug where a timeout occurs when trying to refresh tokens with an already timed-out command.

Key Assertions:

- The `get_token()` method of `TokenRefresher` raises a `TokenRefreshError` exception when the specified command times out.
- The error message 'timed out' is present in the error message returned by the `pytest.raises(TokenRefreshError)` context manager.
- The test asserts that the error message contains the string 'timed out'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	16 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 113-114)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh.py::TestTokenRefresher::test_token_caching` 1ms 3

AI ASSESSMENT

Scenario: Test Token Caching: Verify that the TokenRefresher caches tokens and does not call the command again.

Why Needed: This test prevents a potential bug where the `TokenRefresher` calls the command multiple times if it is caching tokens.

Key Assertions:

- The `token-cached` scenario verifies that the cached token is used instead of calling the command again.
- The `same-token` scenario verifies that the same token is returned even after a successful cache.
- The `only-called-once` scenario verifies that only one call to the `get_token()` method is made by the TokenRefresher.
- The `call_count` assertion verifies that the correct number of calls to the `run()` function are made.
- The `output_format` and `stderr` assertions verify that the output format and error message match the expected values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	28 lines (ranges: 59-60, 63-66, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132, 153-154, 156)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_token_refresh_coverage.py

9 tests

PASSED `tests/test_token_refresh_coverage.py::TestTokenRefresherEdgeCases::test_command_failure_no_stderr` 1ms 3

AI ASSESSMENT

Scenario: Test the TokenRefresher's behavior when a command fails with no stderr output.

Why Needed: Prevent regression that may occur if the test case does not handle command failures properly.

Key Assertions:

- The `get_token()` method of the `TokenRefresher` should raise a `TokenRefreshError` with an exit code of 1 when the command fails without producing any stderr output.
- The error message 'No error output' should be present in the exception message.
- The test case should fail when the command returns an exit code other than 0 or 1.
- The `get_token()` method should not raise a `TokenRefreshError` if the command produces stderr output.
- The `get_token()` method should return an error message indicating that no error occurred.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	20 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 101-104, 113, 115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh_coverage.py::TestTokenRefresherEdgeCases::test_empty_command_string` 1ms 3

AI ASSESSMENT

Scenario: Test handling of empty command string.

Why Needed: Prevents a potential bug where an empty command string is passed to the TokenRefresher, causing it to raise a TokenRefreshError without providing any meaningful error message.

Key Assertions:

- The `get_token()` method raises a `TokenRefreshError` with the message 'empty'.
- The assertion checks if the error message contains the word '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/token_refresh.py	11 lines (ranges: 59-60, 63, 69, 83, 85-86, 90-91, 113, 115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh_coverage.py::TestTokenRefresherEdgeCases::test_invalid_command_string` 1ms 3

AI ASSESSMENT

Scenario: Test handling of invalid command string (shlex parse error).

Why Needed: To prevent a TokenRefreshError caused by an invalid shell syntax in the command string.

Key Assertions:

- The 'command' parameter should be a valid shlex-quoted string.
- Invalid characters or quotes in the 'command' parameter will raise a TokenRefreshError.
- The 'refresh_interval' and 'output_format' parameters are not affected by the 'command' parameter.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	11 lines (ranges: 59-60, 63, 69, 83, 85-88, 113, 115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh_coverage.py::TestTokenRefresherEdgeCases::test_json_not_dict` 1ms 3

AI ASSESSMENT

Scenario: Test that a TokenRefresher fails when the output of get-token is not a dictionary.

Why Needed: Prevents a potential bug where a non-dictionary JSON output causes a TokenRefreshError.

Key Assertions:

- assert 'Expected JSON object' in str(exc_info.value)
- assert 'list' in str(exc_info.value)
- assert 'Invalid JSON format' in str(exc_info.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/token_refresh.py	27 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 101, 107-108, 111, 113, 115, 132-137, 149)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh_coverage.py::TestTokenRefresherEdgeCases::test_json_token_empty_string` 1ms 3

AI ASSESSMENT

Scenario: Test handling when token value is an empty string.

Why Needed: Prevents TokenRefreshError due to incorrect JSON output from subprocess.run().

Key Assertions:

- The 'stdout' of the subprocess.run() call should be a JSON object with a single key-value pair.
- The 'json.dumps()' function should return an empty string if the input is an empty or not a string.
- The error message should contain the phrase 'empty or not a string'.
- The 'stderr' of the subprocess.run() call should be an empty string.
- The 'returncode' of the subprocess.run() call should be 0 (success).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	30 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 101, 107-108, 111, 113, 115, 132-135, 139, 143-146, 149)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh_coverage.py::TestTokenRefresherEdgeCases::test_json_token_not_string` 3ms 3

AI ASSESSMENT

Scenario: Test handling when token value is not a string.

Why Needed: Prevents regression where the TokenRefresher incorrectly handles non-string token values.

Key Assertions:

- assert 'empty or not a string' in str(exc_info.value)
- assert exc_info.value.args[0].startswith('TokenRefreshError')
- assert isinstance(exc_info.value.args[1], str)
- assert json.dumps({'token': 12345}) is None
- assert 'token' not in exc_info.value.args[2]
- assert exc_info.value.args[3] == ''

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	30 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 101, 107-108, 111, 113, 115, 132-135, 139, 143-146, 149)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh_coverage.py::TestTokenRefresherEdgeCases::test_oserror_on_execution` 1ms 3

AI ASSESSMENT

Scenario: Test verifies that a TokenRefresher throws an exception when executing a command that does not exist.

Why Needed: To prevent a potential bug where the TokenRefresher fails to refresh tokens due to a non-existent command.

Key Assertions:

- The `get_token()` method raises a `TokenRefreshError` with a message indicating 'Failed to execute'.
- The `get_token()` method does not raise an exception when executing a nonexistent command.
- The `refresh_interval` is set to 3600 seconds, which may cause issues if the command takes longer than this interval to complete.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	19 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 113, 115-118)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh_coverage.py::TestTokenRefresherEdgeCases::test_text_only_whitespace_lines` 1ms 3

AI ASSESSMENT

Scenario: Test handling when text output has only whitespace lines after initial strip.

Why Needed: Prevents TokenRefreshError due to incorrect parsing of text with only blank lines.

Key Assertions:

- assert 'No non-empty lines' in str(exc_info.value) for exc_info in [pytest.raises(TokenRefreshError)]
- assert len(exc_info.value.args[0].splitlines()) == 1
- assert all(line.strip() == '' for line in exc_info.value.args[0].splitlines())
- assert 'Only whitespace lines' not in exc_info.value.args[0]
- assert 'No non-empty lines' not in exc_info.value.args[0]
- assert len(exc_info.value.args[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/llm/token_refresh.py	4 lines (ranges: 132, 153-155)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED `tests/test_token_refresh_coverage.py::TestTokenRefresherEdgeCases::test_whitespace_only_command` 1ms 3

AI ASSESSMENT

Scenario: Test the test_whitespace_only_command to ensure it raises a TokenRefreshError for an empty command string.

Why Needed: To prevent a potential bug where the token refresh fails due to an empty command string, which could lead to unexpected behavior or errors in the system.

Key Assertions:

- The `get_token()` method of the `TokenRefresher` class should raise a `TokenRefreshError` when given an empty command string.
- The error message returned by the `get_token()` method should indicate that the input was empty.
- The `str(exc_info.value).lower()` assertion should check for the presence of 'empty' in the error message, ensuring it's not just a generic 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/token_refresh.py	11 lines (ranges: 59-60, 63, 69, 83, 85-86, 90-91, 113, 115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)