Test Report

Run ID: 21120216172-py3.12 • Generated: 2026-01-20 01:12:46 • Duration: 34.11s
Plugin: v0.2.0 (b7a157f6cb9189cc50a17c846484c8454deeac61) [dirty]
Repo: v0.2.0 (49c1daf74f0dddf8346192514b7a3d09212b7d87)
LLM: ollama / llama3.2:1b (minimal context, 492 annotated, 39 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 the aggregation of all policy reports to ensure both retained tests are included in the aggregated report.

Why Needed: This test prevents regression where only one or neither of the aggregate reports includes both retained tests.

Key Assertions:

- The aggregated report should include both retained tests from each individual report.
- Both retained tests should be present in the aggregated report.
- No retained tests should be missing from the aggregated report.
- All retained tests should be included in the aggregated report.
- If only one test is retained, it should still be included in the aggregated report.
- If no tests are retained, both aggregate reports should not include any retained tests.
- The aggregated report should have a non-empty list of tests.
- The aggregated report should have at least two unique tests from each individual 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 work correctly when the aggregation directory does not exist.

Key Assertions:

- The `aggregate` method of the aggregator object should return `None` when the specified aggregation directory does not exist.
- The `aggregate` method should raise an exception or handle the situation appropriately if the aggregation directory does not exist.
- The test should fail when the aggregation directory does not exist, indicating a bug in the aggregate function.
- The test should verify that the aggregate function correctly handles the absence of the specified aggregation directory.
- The `aggregate` method should be able to handle cases where the aggregation directory is not found without raising an exception or returning an error message.
- The test should ensure that the aggregator object remains in a valid state after calling the `aggregate` method when the aggregation directory does not exist.

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` 4ms 3

AI ASSESSMENT

Scenario: Test: tests/test_aggregation.py::TestAggregator::test_aggregate_latest_policy

Why Needed: Prevents regression in aggregation of latest policy for different test runs.

Key Assertions:

- The aggregated result should be the latest report's outcome.
- The number of tests passed should match between both reports.
- The run meta should indicate that this is an aggregated run.
- The summary should contain exactly one pass and zero failures.
- The total run count should equal 2 (both runs were run on different times).
- The 'passed' value in the summary should be 1 for report2 (the latest)
- The 'failed' value in the summary should be 0 for report2 (the latest)

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: Test that aggregate function returns None when no directory configuration is provided

Why Needed: Prevents a potential bug where the aggregate function throws an exception or returns incorrect results without specifying a directory.

Key Assertions:

- The `aggregate` method of the `Aggregator` instance should return `None` when called with a mock_config that has no `aggregate_dir` set.
- The `aggregate` method should not throw an exception or raise an error when called with a mock_config that has no `aggregate_dir` set.
- The aggregate function should behave as expected without specifying a directory, i.e., it should return the aggregated data without any modifications.

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 function returns None when no reports exist or are empty.

Why Needed: Prevents regression in case of an empty report directory or a Path.exists check failing.

Key Assertions:

- The aggregator.aggregate() method should return None for the given input.
- The aggregate function should not be able to find any reports when there are no reports or empty directories.
- The aggregate function should raise an exception when it cannot find any reports in the directory.
- The test should fail with a ValueError if the Path.exists check returns False but the Path.glob check returns an empty list.
- The test should pass with a None value for the aggregator.aggregate() method.

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

LLM error: Failed to parse LLM response as JSON

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 source coverage summary is deserialized correctly.

Why Needed: Prevents regression in aggregate calculation when source coverage data is missing or incomplete.

Key Assertions:

- The 'source_coverage' key in the report should be an array of SourceCoverageEntry objects.
- Each SourceCoverageEntry object should have the following properties: file_path, statements, missed, covered, coverage_percent, and covered_ranges.
- The 'coverage_percent' property should be a number between 0 and 100.
- The 'covered_ranges' property should contain strings representing ranges of covered lines (e.g., '1-5, 7-11').
- The 'missed_ranges' property should contain strings representing ranges of missed lines (e.g., '6, 12').
- Each SourceCoverageEntry object should have a 'file_path' attribute matching the file path in the report.
- The 'source_coverage' array should only contain one element when deserialization is successful.
- The 'source_coverage' array should be an instance of SourceCoverageEntry.

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` 4ms 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

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 aggregator recalculates summary with given tests and latest summary.

Why Needed: This test prevents regression in the aggregator's recalculate_summary method, ensuring it correctly handles incomplete summaries.

Key Assertions:

- The total number of tests passed is equal to the expected value (6).
- The number of passed tests is equal to the expected value (1).
- The number of failed tests is equal to the expected value (1).
- The number of skipped tests is equal to the expected value (1).
- The number of xfailed tests is equal to the expected value (1).
- The number of xpassed tests is equal to the expected value (1).
- The number of error tests is equal to the expected value (1).

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 that skipping an invalid JSON file prevents a regression in aggregation.

Why Needed: Regression test to ensure skipping an invalid JSON file does not affect the overall aggregation functionality.

Key Assertions:

- - The `aggregate` function skips the invalid JSON report file.
- - Only valid reports are counted in the aggregated output.
- - Skipping an invalid JSON file should raise a warning with a specific message.
- - The test verifies that the `aggregate` function does not return any result when skipping an invalid JSON file.
- - The test verifies that the test suite still runs successfully without errors.
- - Skipping an invalid JSON file is done before running other tests in the suite.

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 durations.

Why Needed: This test prevents regression in the aggregator's behavior when handling different test durations.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/aggregation.py	10 lines (ranges: 44, 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` 2ms 5

AI ASSESSMENT

Scenario: Verify that cached tests are skipped by the annotator.

Why Needed: This test prevents a regression where the annotator incorrectly includes cached tests in the analysis.

Key Assertions:

- Mocked `mock_provider` is not called with any arguments.
- Mocked `mock_cache` is not called to retrieve test data.
- Mocked `mock_assembler` is not called with any arguments.
- The `test_cached_tests_are_skipped` function does not call `mock_provider` before returning.
- The `test_cached_tests_are_skipped` function does not call `mock_cache` after returning.
- The `test_cached_tests_are_skipped` function does not call `mock_assembler` after returning.
- Mocked `mock_provider` is called with an argument (e.g., a test class) but the annotation is skipped.
- Mocked `mock_cache` is called to retrieve test data but the annotation is skipped.
- Mocked `mock_assembler` is not called at all.

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 `test_concurrent_annotation` function is being tested to ensure that annotators can annotate data concurrently without causing any issues.

Why Needed: This test prevents potential performance regressions or bugs caused by concurrent annotation in the presence of multiple annotators.

Key Assertions:

- mock_provider should not raise an exception when annotated concurrently
- mock_cache should not be modified while being accessed concurrently
- mock_assembler should not raise an exception when annotated concurrently

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 handles failures correctly.

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

Key Assertions:

- Mocking the provider, cache, and assembler with mock objects.
- Capturing the output of the annotator using capsys.
- Asserting that the annotator raises an exception when it encounters a failure.
- Verifying that the annotator handles failures by logging the error message.
- Checking if the annotator correctly returns a success status for successful annotations.
- Ensuring that the annotator does not crash or raise an exception unexpectedly in concurrent scenarios.

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: The test verifies that the progress reporting functionality is working correctly.

Why Needed: This test prevents regressions where the progress reporting might not be accurate or complete.

Key Assertions:

- Mocked progress bar should update in real-time as data is processed.
- Mocked progress bar should display correct percentage and total progress.
- Mocked progress bar should update correctly when data is loaded from cache.
- Mocked progress bar should not update if no data is available for processing.
- Mocked progress bar should reset to 0% after completion of all tasks.
- Mocked progress bar should display correct units (e.g., MB, KB, etc.) for large files.
- Mocked progress bar should handle errors and exceptions properly when loading from cache.

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 functionality of the annotator.

Why Needed: Prevents regression in sequential annotation workflow.

Key Assertions:

- The function should be able to annotate data sequentially without any issues.
- The annotator should not return an error if the annotation sequence is invalid.
- The cache should be updated correctly when the annotation sequence changes.
- Mock providers, assemblers, and caches should be mocked with the correct dependencies.
- The annotator's output should match the expected annotations in each step of the sequential process.

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

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/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

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/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: Test that annotator reports progress and first error when annotated concurrently with progress and errors.

Why Needed: This test prevents regression by ensuring that the annotator correctly handles concurrent annotations with progress and errors.

Key Assertions:

- Verify that the annotator reports the correct number of tasks (2) and failures (1) after concurrent annotation.
- Check if the first error is included in the annotated results.
- Verify that a processing message is appended to the list of progress messages.
- Ensure that at least one LLM annotation message is included in the list of progress messages.
- Confirm that the annotator correctly reports the LLM annotation for the first 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/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: Should wait if rate limit interval has not elapsed.

Why Needed: Prevents regression where the annotator does not wait for the rate limit interval to pass before proceeding with annotation tasks.

Key Assertions:

- assert mock_sleep.called
- assert mock_time.call_count == 5
- assert mock_time.side_effect == [100.0, 100.1, 100.2, 100.3, 100.4]
- assert tasks[0].outcome == 'p' and tasks[0].nodeid == 't1'
- assert tasks[1].outcome == 'p' and tasks[1].nodeid == 't2'
- assert tasks[2].outcome == 'p' and tasks[2].nodeid == 't3'
- assert tasks[3].outcome == 'p' and tasks[3].nodeid == 't4'
- assert tasks[4].outcome == 'p' and tasks[4].nodeid == 't5'

COVERAGE

src/pytest_llm_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: Test that the annotator reports progress when caching tests.

Why Needed: This test prevents regression where the annotator does not report progress for cached tests.

Key Assertions:

- The 'cache' key should be present in the progress messages.
- Each message should contain the string '(cache): test_cached'.
- The annotator should return a non-empty list when calling `append` on `progress_msgs`.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	37 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-84, 97-98, 100, 127, 129-135, 137, 139)
src/pytest_llm_report/options.py	3 lines (ranges: 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` 2ms 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/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

Scenario: The test verifies that the `test_base_parse_response_malformed_json_after_extract` function ensures that an error is raised when a malformed JSON is extracted from a response.

Why Needed: This test prevents a potential regression where a malformed JSON might be successfully parsed, potentially leading to incorrect LLM response interpretation.

Key Assertions:

- The `annotation.error` attribute will contain the string 'Failed to parse LLM response as JSON'.
- The `provider._parse_response(response)` call will raise a `JSONDecodeError` exception.
- The `annotation.error` attribute will not be empty after the error is raised.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: Verify that the `_parse_response` function correctly handles non-string fields in a response data dictionary.

Why Needed: To prevent bugs and regressions where the function incorrectly assumes all fields are strings, especially when dealing with non-standard or external data sources.

Key Assertions:

- The `scenario` attribute of the annotation is set to the correct value (123).
- The `why_needed` attribute of the annotation is set to a list containing 'list'.
- The `key_assertions` attribute of the annotation is set to the correct value ('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 valid instance of `GeminiProvider`.

Why Needed: This test prevents a potential bug where the `get_gemini_provider` function may return an incorrect or null provider.

Key Assertions:

- The returned value is indeed an instance of `GeminiProvider`.
- The provider is not null.
- The provider has the expected attributes (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	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: Testing the `get_invalid_provider` method of the `Config` class to ensure it raises a ValueError when an unknown LLM provider is specified.

Why Needed: To prevent a potential bug where an unknown LLM provider is attempted without raising an error, and instead, to provide clear feedback to the user that an invalid provider was used.

Key Assertions:

- The `get_invalid_provider` method should raise a `ValueError` with a message indicating that the specified provider is unknown.
- The `match` parameter in the `pytest.raises` context match should contain the string 'Unknown LLM provider: invalid'.
- The `config` variable passed to the `test_get_invalid_provider` function should be an instance of `Config` with a `provider` attribute set to 'invalid'.
- The `get_provider(config)` call within the test should raise a `ValueError` exception.
- The `pytest.raises` context match should contain the string 'Unknown LLM provider: invalid' as the error message.
- The `config` variable passed to the `test_get_invalid_provider` function should be an instance of `Config` with a `provider` attribute set to 'invalid'.
- The `get_provider(config)` call within the test should raise a `ValueError` exception when called with an invalid provider.
- A `ValueError` exception should be raised when calling `get_provider(config)`, indicating that the specified LLM provider is unknown.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 245, 247, 249, 252, 257, 262, 267)
src/pytest_llm_report/options.py	2 lines (ranges: 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: Verifies that the `get_litellm_provider` function returns an instance of `LiteLLMProvider` when a valid configuration is provided.

Why Needed: This test prevents a potential bug where the `get_litellm_provider` function may return an incorrect or invalid provider instance if the configuration is not properly validated.

Key Assertions:

- The function `get_provider(config)` returns an instance of `LiteLLMProvider` when a valid `Config` object is passed.
- The returned `LiteLLMProvider` instance has the correct attributes and methods.
- The `LiteLLMProvider` instance can be instantiated with the provided configuration.
- The `LiteLLMProvider` instance does not have any unexpected or invalid attributes.
- The `LiteLLMProvider` instance has all required methods (e.g. `__init__`, `get_config`, etc.) implemented correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: Tests the functionality of getting a provider with no configuration.

Why Needed: Prevents a potential bug where a provider is created without any configuration, potentially leading to unexpected behavior or errors.

Key Assertions:

- The `get_provider` function should return an instance of `NoopProvider` when given no configuration.
- The `provider` attribute of the returned `NoopProvider` instance should be `None`.
- The `config` parameter passed to `get_provider` should not affect the creation of a `NoopProvider` instance.
- An error message or indication that the provider was created with no configuration should be provided by the `get_provider` function.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 regression in case the `provider` is set to 'ollama' and the function fails to return a valid provider.

Key Assertions:

- The returned value is an instance of `OllamaProvider`.
- The `provider` parameter matches `'ollama'`.
- The `get_provider` function returns a valid provider.
- The `provider` attribute of the returned provider is set to 'ollama'.
- The `provider` attribute of the returned provider has a value that does not match any known providers.
- The `provider` attribute of the returned provider is an instance of `OllamaProvider`.
- The `provider` attribute of the returned provider has a type that matches the expected type.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 245, 247, 249, 252-253, 255)
src/pytest_llm_report/options.py	2 lines (ranges: 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 LLM provider returns a boolean indicating availability and a non-zero checks counter.

Why Needed: This test prevents regression in case the LLM provider implementation changes, causing it to return False for available caches without incrementing the checks counter.

Key Assertions:

- provider.is_available() is True
- provider.is_available() is True
- provider.checks == 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/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 if no custom model is provided in the configuration.

COVERAGE

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

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	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: Verifies that `is_local()` returns `False` when the LLM defaults to local mode.

Why Needed: Prevents regression in case of changes to LLM defaults or configuration.

Key Assertions:

- The `provider.is_local()` method should return `False` for default LLM configurations.
- The `provider.is_local()` method should not raise an exception when the default LLM configuration is used.
- The `is_local()` method should be able to handle different types of LLM providers (e.g., `ConcreteProvider`, `Text2ModelProvider`, etc.).
- The `is_local()` method should work correctly even if the LLM defaults are not explicitly set in the configuration.
- The `provider.is_local()` method should return `False` for default LLM configurations with specific settings (e.g., `max_length=512`).

COVERAGE

src/pytest_llm_report/collector.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: Testing the consistency of a cache with the same source function.

Why Needed: Prevents a potential bug where different functions producing the same source code could produce different hashes, leading to inconsistent caching behavior.

Key Assertions:

- The hash returned for the given source should be equal to the hash returned for the same source.
- The hash of the function body 'def test_foo(): pass' should be equal to itself.
- The cache should store the result of the function call with the given source.
- The cache should not store any other results from different functions producing the same source code.
- The cache should maintain consistency across multiple function calls with the same source.
- The hash of the source code 'def test_foo(): pass' should be unique and not equal to any other source code.
- The cache should not store any cached results for the same source function in different environments (e.g., different Python versions).

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

Scenario: Verifies that the `hash_source` function returns a different hash for two different source code strings.

Why Needed: Prevents a bug where the same source code string produces the same hash value, potentially leading to unexpected behavior or incorrect results in cache-based algorithms.

Key Assertions:

- The `hash_source` function should return a different hash value for `def test_a(): pass` and `def test_b(): pass`.
- The `hash_source` function should not produce the same hash value for two source code strings that are likely to be different (e.g., different functions, variables, etc.).
- The `hash_source` function should raise an error or return a meaningful value when given duplicate source code strings.
- The `hash_source` function should use a consistent and predictable algorithm for generating hash values.
- The `hash_source` function should not be sensitive to the order of operations within the source code string (e.g., parentheses, semicolons, etc.).
- The `hash_source` function should handle edge cases where the source code string is empty or contains only whitespace characters.
- The `hash_source` function should use a consistent naming convention for functions and variables in the source code strings.
- The `hash_source` function should not produce the same hash value for source code strings that contain different types of data (e.g., integers, floats, strings, etc.).

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: Prevent a potential issue where the hash is too short, potentially leading to incorrect caching or storage.

Key Assertions:

- The length of the hash should be exactly 16 characters.
- The hash should not be shorter than 16 characters.
- The hash should not exceed 32 characters (the maximum allowed by Python).
- The hash should start with a hexadecimal digit (0-9, A-F).
- The hash should contain only hexadecimal digits (0-9, A-F, a-f).
- The hash should be an empty string if the input is not a valid hash source.
- The hash should raise a `TypeError` if the input is not a hashable object.

COVERAGE

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

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

AI ASSESSMENT

Scenario: Test that clearing the cache removes all entries.

Why Needed: Prevents a bug where multiple cache entries are left after clearing, potentially causing unexpected behavior in downstream code.

Key Assertions:

- Verify that only two cache entries exist before and after clearing.
- Ensure that both 'test::a' and 'test::b' are removed from the cache after clearing.
- Check that attempting to retrieve a non-existent key ('test::c', 'hash3') returns None for both keys.
- Verify that the number of cache entries is correctly decremented by 2 during clearing.
- Confirm that the cache directory remains empty after clearing, regardless of the number of entries.

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: To prevent caching of error-related annotations, ensuring that non-cachable annotations do not interfere with the test environment.

Key Assertions:

- The annotation 'error' is set to a string value.
- The cache does not store the annotation 'test::foo' for the key 'test::foo'.
- The annotation 'abc123' is retrieved from the cache without error.
- The result of the retrieval operation is None, indicating that caching was disabled.
- No exception is raised during the execution of the test.

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 that `get` method returns None for missing entries in the cache.

Why Needed: Prevents a potential bug where an empty string is returned when trying to access a non-existent key in the cache.

Key Assertions:

- The function should return `None` when the specified key does not exist in the cache.
- The function should raise a `KeyError` exception if the specified key does not exist in the cache.
- The function should handle missing keys by returning `None` instead of raising an error or returning an empty string.
- The function should be able to handle cases where the cache is initialized with a non-empty directory path.
- The function should correctly handle cases where the specified key is not found in the cache even if it exists elsewhere.
- The function should raise an exception when trying to access a missing key that does not exist in the cache.

COVERAGE

src/pytest_llm_report/cache.py	9 lines (ranges: 39-41, 53, 55-56, 118-119, 121)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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 the functionality of storing and retrieving annotations in the LLMCache.

Why Needed: Prevents bypass attacks by ensuring that LLMCache stores and retrieves annotations securely.

Key Assertions:

- Verify that the annotation is stored correctly in the cache with the given key.
- Check if the retrieved annotation matches the expected value.
- Ensure that the confidence level of the retrieved annotation matches the expected value.
- Verify that the annotation is not empty or None when retrieved.
- Check if the annotation's scenario and confidence match the original values.
- Confirm that the cache stores annotations securely by checking for any errors.

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 collection error has the correct node ID and message.

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

Key Assertions:

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

COVERAGE

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

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

AI ASSESSMENT

Scenario: Verifies that the `get_collection_errors` method returns an empty list when the collection is initially empty.

Why Needed: Prevents a potential bug where the test fails with an error message indicating that there are no errors in an empty collection.

Key Assertions:

- asserts that the `get_collection_errors()` method returns an empty list
- asserts that the `get_collection_errors` method does not raise any exceptions when called on an empty collection
- asserts that the `get_collection_errors` method correctly identifies no errors in an empty collection

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: The test verifies that the default llm_context_override is set to None for a TestCaseResult.

Why Needed: This test prevents a potential bug where the default llm_context_override is not correctly set to None, potentially leading to incorrect results or unexpected behavior in downstream tests.

Key Assertions:

- The llm_context_override attribute of the TestCaseResult object should be None.
- The llm_context_override attribute of the TestCaseResult object should not be explicitly set by the test.
- The default value for llm_context_override is correctly set to None in the TestCaseResult model.
- The llm_context_override attribute of a TestCaseResult object with an outcome of 'passed' should still be None.
- A TestCaseResult object with an outcome of 'failed' or any other non-'passed' outcome should have an llm_context_override attribute set to None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 the default value of llm_opt_out for LLM optimization out.

Why Needed: Prevents regression when default llm_opt_out is set to False.

Key Assertions:

- The llm_opt_out attribute should be initialized with a default value of False.
- A TestCaseResult object created with an empty nodeid and outcome should have llm_opt_out as True.
- Setting llm_opt_out to False in the TestCaseResult object should not affect the test result.
- The TestCaseResult object's llm_opt_out attribute should be initialized correctly even if it is not set explicitly.
- The TestCaseResult object's llm_opt_out attribute should remain False after the test execution.
- The TestCaseResult object's llm_opt_out attribute should be False when the nodeid is empty or None.
- The TestCaseResult object's llm_opt_out attribute should be False when the outcome is 'passed' and default llm_opt_out is True.
- The TestCaseResult object's llm_opt_out attribute should not be affected by the pytest_llm_report.models(TestCaseResult) constructor.

COVERAGE

src/pytest_llm_report/collector.py	14 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` option in the configuration is set to `False` by default.

Why Needed: Without this test, the `capture` option may not be properly configured or disabled as intended.

Key Assertions:

- assert config.capture_failed_output is False
- assert isinstance(config, Config)
- assert hasattr(config, 'capture') and config.capture == False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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' in the 'Config' class is 4000.

Why Needed: This test prevents a potential bug where the default max chars for capturing output is not set to 4000, potentially leading to issues with large files or long-running tests.

Key Assertions:

- The value of 'capture_output_max_chars' in the 'Config' class should be 4000.
- The maximum number of characters that can be captured as output without exceeding the default value of 4000 is 4000.
- The configuration object does not have a 'capture_output_max_chars' attribute.
- The 'capture_output_max_chars' attribute in the 'Config' class is not being used or updated correctly.
- Setting the 'capture_output_max_chars' to a different value than 4000 could cause unexpected behavior in tests that rely on this 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	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' verifies that when an XFAIL is encountered, it will be recorded as xfailed in the test results.

Why Needed: This test prevents regression by ensuring that XFAILs are correctly recorded and reported to the user.

Key Assertions:

- The 'xfail' flag is set on the report object.
- The 'wasxfail' field of the report object indicates whether an XFAIL was encountered.
- The outcome of the test result is set to 'xfailed'.
- The duration and longrepr fields are correctly initialized for the report object.
- The 'passed', 'skipped', and 'longrepr' fields are all set to False for the reported runtest log report.
- The 'wasxfail' field is set to 'expected failure' for the reported runtest log report.
- The outcome of the test result is correctly set to 'xfailed'.

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: The test verifies that xfail passes are correctly recorded as xpassed.

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

Key Assertions:

- The `result.outcome` of the runtest log report should be 'xpassed'.
- The `wasxfail` field of the runtest log report should contain the string 'expected failure'.
- The `duration` field of the runtest log report should be 0.01 seconds.
- The `skipped` field of the runtest log report should be False.
- The `failed` field of the runtest log report should be False.
- The `passed` field of the runtest log report should be True.

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 that the `TestCollector` class initializes correctly with an empty collection.

Why Needed: This test prevents a potential bug where the `TestCollector` class does not initialize properly, potentially causing incorrect results or behavior in subsequent tests.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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 a regression where the results are not sorted correctly due to manual modifications.

Key Assertions:

- The list of node IDs should be in ascending order.
- The list of node IDs should contain all the expected nodes.
- The node ID 'a_test.py::test_a' should come first in the list.
- The node ID 'z_test.py::test_z' should come second in the list.
- No other nodes should be present in the sorted list.
- All node IDs should have a valid outcome ('passed').
- The `nodeid` attribute of each result object should match the expected values.

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

Why Needed: This test prevents a potential regression where the count of collected items is not updated correctly after finishing collection.

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 if config disabled (integration via handle_runtest_logreport)' verifies that the collector does not capture output when configured to disable capturing.

Why Needed: This test prevents a regression where the collector captures output even though the configuration is set to disable it, which could lead to unexpected behavior or errors in downstream tests.

Key Assertions:

- The 'captured_stdout' attribute of the report node should be None after calling `handle_runtest_logreport` on the collector with a report that was created without capturing output.
- The 'wasxfail' attribute of the report node should not be set to True if the report was created without capturing output.
- The 'outcome' attribute of the report node should be set to 'failed' after calling `handle_runtest_logreport` on the collector with a report that failed.
- The 'when' attribute of the report node should be set to 'call' after calling `handle_runtest_logreport` on the collector with a report that was created without capturing output.
- The 'nodeid' attribute of the report node should be set to 't' after calling `handle_runtest_logreport` on the collector with a report that failed.
- The 'passed' attribute of the report node should be False if the report was created without capturing output.
- The 'failed' attribute of the report node should be True if the report was created without capturing output and the outcome is 'failed'.
- The 'skipped' attribute of the report node should be False if the report was created without capturing output.

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 `test_capture_output_stderr` function captures stderr correctly.

Why Needed: This test prevents a potential bug where the `test_capture_output_stderr` function does not capture stderr if an error occurs during collection.

Key Assertions:

- The captured stderr should be 'Some error'.
- The report should have been able to capture and display the stderr message.
- The `report.capstderr` method should have been called with a string argument containing 'Some error'.

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

LLM error: Failed to parse LLM response as JSON

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 captured stdout exceeds the maximum characters set in the config, potentially causing unexpected behavior or errors.

Key Assertions:

- The captured stdout should be truncated to 10 characters.
- The captured stderr should not have any characters.
- The `captured_stdout` attribute of the `TestCaseResult` object should contain only '1234567890'.
- The `report.capstdout` attribute should contain the expected string.
- The `report.capstderr` attribute should be an empty string.
- The `_capture_output` method should call the `captured_stdout` attribute of the `TestCaseResult` object with the truncated output.
- The captured stdout and stderr should not exceed the maximum characters set in the config.

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 `test_create_result_with_item_markers` verifies that the collector extracts item markers correctly.

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

Key Assertions:

- item.get_closest_marker('llm_opt_out') returns MagicMock().
- item.get_closest_marker('llm_context_override') returns MagicMock().
- item.get_closest_marker('requirement') returns MagicMock().
- result.param_id is set to 'param1' as expected.
- result.llm_opt_out is set to True as expected.
- result.llm_context_override is set to 'complete' as expected.
- result.requirements contains the correct list of strings ['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` 2ms 3

AI ASSESSMENT

Scenario: Test the `test_extract_error_repr_crash` function to ensure it handles ReprFileLocation correctly.

Why Needed: This test prevents a potential crash in the `test_extract_error_repr_crash` function when using ReprFileLocation.

Key Assertions:

- The `_extract_error` method of `TestCollector` should return 'Crash report' when given a `Report` object with a `longrepr` attribute that is set to a string value.
- The `__str__` method of the `longrepr` object should be called and its return value should be equal to 'Crash report'.
- The `report.longrepr.__str__.return_value` property should be accessed and its value should match the expected output.

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 the `_extract_error` method of `TestCollector` to ensure it returns a string that is directly related to the error.

Why Needed: The current implementation does not handle cases where the error message contains long strings, potentially leading to incorrect results or errors in downstream processing.

Key Assertions:

- assert isinstance(collector._extract_error(report), str)
- assert collector._extract_error(report) == 'Some error occurred'
- assert report.longrepr is None
- assert len(collector._extract_error(report)) > 0

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 of `TestCollector` when no longrepr is provided.

Why Needed: Prevents a potential bug where the test fails to extract skip reasons due to an empty longrepr value.

Key Assertions:

- The `report.longrepr` attribute is set to `None` before calling `_extract_skip_reason`.
- The `_extract_skip_reason` method returns `None` when no longrepr is provided.
- The test asserts that the returned value is indeed `None` for the given scenario.

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: The test verifies that the `test_extract_skip_reason_string` method returns a string 'Just skipped' as skip reason.

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

Key Assertions:

- report.longrepr is set to 'Just skipped'
- assert collector._extract_skip_reason(report) == 'Just skipped'
- The method returns a string that is exactly 'Just skipped'
- The method does not raise an exception when the report is empty or None

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 `collectors._extract_skip_reason` returns a string representation of the skip message from a tuple.

Why Needed: This test prevents a potential issue where the `collectors._extract_skip_reason` function does not return a string representation of the skip message, potentially leading to unexpected behavior or errors when trying to compare it with other values.

Key Assertions:

- The `report.longrepr` attribute should contain a tuple with three elements: `(file, line, message)`.
- The `message` element in the tuple should be a string.
- The `message` element should not be empty or None.
- The `message` element should match the expected reason string for skipping the test.
- The `report.longrepr` attribute should contain at least one non-empty string value.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250-251)
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: When the `handle_collection_report` method is called with a report that indicates a collection error, it should record this error in the `collection_errors` list.

Why Needed: This test prevents a potential bug where a collection report failure is not properly recorded and reported to the user, potentially leading to confusion or incorrect reporting of errors.

Key Assertions:

- The length of `collector.collection_errors` should be equal to 1.
- The nodeid in `collector.collection_errors[0]` should match 'test_broken.py'.
- The message in `collector.collection_errors[0]` should match 'SyntaxError'.

COVERAGE

src/pytest_llm_report/collector.py	21 lines (ranges: 58, 60-65, 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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` 2ms 3

AI ASSESSMENT

Scenario: The test verifies that the `handle_runtest_rerun` method of the `TestCollector` class correctly handles reruns by setting the `rerun` attribute of the report to 1 and updating the final outcome to 'failed'.

Why Needed: This test prevents a regression where the `rerun` attribute is not set correctly in some cases, potentially leading to incorrect reporting.

Key Assertions:

- The `rerun_count` property of the `results` dictionary should be equal to 1.
- The `final_outcome` property of the `results` dictionary should be equal to 'failed'.
- The `wasxfail` attribute is removed from the report after handling rerun.
- The `nodeid` and `when` attributes are set correctly on the report before passing it to the `handle_runtest_logreport` method.
- The `rerun` attribute of the report is updated correctly to 1.
- The final outcome of the test is 'failed' after rerunning the test.
- The `results` dictionary contains a single entry with key `t::r` and value as an object.

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: TestCollectorReportHandling::test_handle_runtest_setup_failure verifies that a setup failure is recorded and handled correctly.

Why Needed: This test prevents the regression of handling setup failures by recording them in the report.

Key Assertions:

- The `report` object has been set to 'setup' when the setup failed.
- The `outcome` attribute of the `res` dictionary is set to 'error'.
- The `phase` attribute of the `res` dictionary is set to 'setup'.
- The `error_message` attribute of the `res` dictionary contains the string '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: When the teardown fails after a pass, then the report should record an error.

Why Needed: The test prevents the collector from silently ignoring the failure and reporting only the success.

Key Assertions:

- res.outcome == 'error'
- res.phase == 'teardown'
- res.error_message == '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: Verify that the GeminiProvider correctly handles edge cases during model parsing.

Why Needed: This test prevents a potential bug where the parser fails to parse models with an empty list, causing incorrect assertions.

Key Assertions:

- The function provider._parse_preferred_models() should return an empty list when the 'model' parameter is None.
- The function provider._parse_preferred_models() should return an empty list when the 'model' parameter is set to 'All'.
- The function provider._parse_preferred_models() should not throw any exceptions when the 'model' parameter is None or 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: Verify that the rate limiter does not allow tokens to be recorded when there are no requests.

Why Needed: This test prevents a potential bug where the rate limiter allows tokens to be recorded even if there are no requests.

Key Assertions:

- assert limiter.next_available_in(60) > 0
- assert limiter.next_available_in(10) == 0
- limiter.record_tokens(50) should not trigger the rate limiter

COVERAGE

src/pytest_llm_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` returns the correct coverage percentage.

Why Needed: Prevents regression in coverage calculation for SourceCoverageEntry model variants.

Key Assertions:

- The value of `d['coverage_percent']` is equal to 50.0.
- The error message from `ann.to_dict()` matches the expected string 'timeout'.
- The duration of `meta.to_dict()` matches the expected value 1.0.
- The coverage range for `sce.to_dict()` matches the expected format '1-5'.
- The coverage range for `meta.to_dict()` matches the expected format '6-10'.

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`	`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: Test that a new `CoverageMapper` instance initializes correctly with the provided configuration.

Why Needed: This test prevents a potential bug where a new `CoverageMapper` instance created from an empty configuration does not initialize properly.

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: Test should return a list of warnings from the coverage map.

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

Key Assertions:

- The `get_warnings()` method of the `CoverageMapper` class is expected to return a list.
- The `warnings` variable is expected to be a list.
- The `assert isinstance(warnings, list)` line checks if the returned value is indeed a 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	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` 2ms 5

AI ASSESSMENT

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

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

Key Assertions:

- The `Path.exists` mock returns False.
- The `glob.glob` mock returns an empty list.
- The `map_coverage` method returns an empty dictionary.
- At least one warning is emitted by the `map_coverage` method.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/coverage_map.py`	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: The test verifies that the `CoverageMapper` can extract node IDs from all phases of a coverage report.

Why Needed: This test prevents regression by ensuring that the `CoverageMapper` correctly extracts node IDs for all phases, even when `include_phase=all` is used.

Key Assertions:

- The extracted node ID matches the expected value for each phase.
- The same node ID is extracted for multiple phases (e.g., `test.py::test_foo|setup` and `test.py::test_foo|teardown`).
- No other nodes are included in the coverage report when `include_phase=all`.
- The test passes even if there are no tests or functions defined in the specified phase.
- The node ID is correctly extracted from the file path (e.g., `test.py::test_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/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

Scenario: Verify that an empty context does not affect the extraction of node IDs.

Why Needed: Prevents a potential bug where an empty context might cause the coverage mapping to fail or produce incorrect results.

Key Assertions:

- The function _extract_nodeid() returns None when given an empty string as input.
- The function _extract_nodeid() returns None when given an empty integer (None) as input.
- The function _extract_nodeid() does not raise any exceptions when given an empty context.
- The function _extract_nodeid() correctly handles non-integer inputs by returning None.
- The function _extract_nodeid() maintains the integrity of the test output even with empty inputs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 potential bug where the coverage map includes setup phases for certain functions, which can lead to inaccurate coverage reports.

Key Assertions:

- The function mapper._extract_nodeid() should return None for nodeids containing 'test.py::test_foo|setup'.
- The include_phase parameter in the Config object should be set to 'run' to exclude setup phases from the coverage map.
- The mapper._extract_nodeid() method should not raise an exception when called with a nodeid containing 'test.py::test_foo|setup'.

COVERAGE

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

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	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 the scenario of handling exceptions when calling `contexts_by_lineno` method.

Why Needed: This test prevents a potential regression where the `contexts_by_lineno` method raises an exception when encountering files with no contexts.

Key Assertions:

- The function should return an empty dictionary when called on a file without any contexts.
- The function should not raise an exception when called on a file with no contexts.
- The function should handle the exception correctly and do not terminate the test execution.
- The function should log the exception properly to avoid masking its details.
- The function should return the expected result for files without any contexts.
- The function should not raise an exception when called on a file with no contexts, allowing it to continue executing.

COVERAGE

src/pytest_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

Scenario: Test Extracting Contexts when no measured files are present.

Why Needed: Prevents regression in coverage data analysis when there are no measured files.

Key Assertions:

- The function `_extract_contexts` is called with a mock object `mock_data` that has an empty list of measured files.
- The expected result for the mocked `mock_data` is an empty dictionary `{}`.
- The `measured_files` attribute of the `mock_data` object is not accessed or modified 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/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:

- The `mock_data.measured_files` call returns the expected list of file names: ['file.txt', 'data.json']
- The `mock_data.contexts_by_lineno` call returns an empty dictionary for all lines
- The `_extract_contexts` method is called with the mock data and asserts it returns 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_coverage.py::TestLoadCoverageData::test_coverage_not_installed` 1ms 4

AI ASSESSMENT

Scenario: Test that coverage.py is properly loaded when it's not installed.

Why Needed: Prevents a potential bug where coverage data is not collected due to missing coverage.py.

Key Assertions:

- The _load_coverage_data method of the CoverageMapper class should return None if coverage.py is not installed.
- The Config object created by the config = Config() line should be None if coverage.py is not installed.
- The mapper variable assigned to mapper = CoverageMapper(config) should be None if coverage.py is not installed.
- The assert statement asserting that mapper is not None at the end of the test should fail if coverage.py is not installed.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 verifies that when no .coverage file exists, the function returns None and generates warnings.

Why Needed: This test prevents a potential regression where the test fails to report coverage data if there is no .coverage file present.

Key Assertions:

- The function `mapper._load_coverage_data()` should return `None` when no `.coverage` file exists.
- A warning with code 'W001' should be generated for each line in the source code.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	9 lines (ranges: 44-45, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 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 that Analysis2 raises an exception and handles it correctly.

Why Needed: This test prevents the regression where Analysis2's analysis fails and no warnings or errors are reported.

Key Assertions:

- The function `map_source_coverage` should return an empty list when `analysis2` raises an exception.
- A warning message should be added to the warnings list for each file that was measured during coverage analysis.
- Any error messages from Analysis2 should be caught by the `Analysis2Exception` and reported as 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	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` 2ms 5

AI ASSESSMENT

Scenario: Test that the test_empty_statements function correctly handles an empty file.

Why Needed: Prevents a regression where coverage is not reported for files with no statements.

Key Assertions:

- The `map_source_coverage` method should return an empty list when given a mock data object with no measured files.
- The `get_data` method of the mock data object should be called without returning any values.
- The `analysis2` method of the mock cov object should be called with no arguments and return the expected result (empty string).
- The `map_source_coverage` method should not raise an exception when given a mock data object with no measured files.
- The `map_source_coverage` method should correctly handle the case where the input is empty.
- The `map_source_coverage` method should report 0 coverage for the empty file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	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 `omit_tests_from_coverage` configuration flag is set to True without providing a valid coverage map.

Key Assertions:

- mocked get_data method returns a single mocked data object with one measured file.
- mocked analysis2 method returns a tuple containing two values: the number of covered files and an empty list.
- mocked analyze method returns a list containing three values: the number of missed files, an empty list, and 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	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` 2ms 4

AI ASSESSMENT

Scenario: Test that non-Python files are skipped.

Why Needed: Prevents regression by skipping non-python files from coverage analysis.

Key Assertions:

- mocked 'file.txt' is not included in the measured files.
- mocked 'data.json' is not included in the measured files.
- The mocked contexts are correctly returned without line numbers.
- No non-Python file is reported as a source of coverage.
- Mocked data does not contain any Python files.
- The mock data's contexts_by_lineno returns an empty dictionary.
- The mock_data.measured_files returns the expected list of measured 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	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 potential regression where the coverage map is not generated correctly when omitting test files from coverage.

Key Assertions:

- The `mock_data.measured_files` mock returns an empty list when `omit_tests_from_coverage=True`.
- The `mock_cov.get_data` method returns the expected `mock_data` object when `omit_tests_from_coverage=True`.
- The `mapper.map_source_coverage` function correctly skips test files and generates a coverage map without any test files.
- The coverage map is generated successfully even if no test files are present in the repository.
- The `mapper.map_source_coverage` function does not throw an exception or raise an error when skipping test files.
- The coverage map includes all measured files, including those that were intended to be skipped.
- The `mock_cov.get_data` method returns a mock object with the expected data structure and attributes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 where only specific phases are covered by the configuration.

Key Assertions:

- The mapper should return nodeid for any phase.
- The mapper should match the expected nodeids for each phase.
- No unexpected nodeids should be returned for any phase.

COVERAGE

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

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_coverage.py::TestPhaseFiltering::test_extract_nodeid_none` 1ms 4

AI ASSESSMENT

Scenario: The test verifies that the `extract_nodeid` method of the `CoverageMapper` class returns `None` when given a `None` input.

Why Needed: This test prevents a potential bug where the `extract_nodeid` method does not correctly handle `None` inputs, potentially leading to incorrect coverage results or unexpected behavior.

Key Assertions:

- The `_extract_nodeid` method of the `CoverageMapper` class should return `None` when given a `None` input.
- The `extract_nodeid(None)` expression should evaluate to `None`.
- A `None` input should not cause any exceptions or errors in the program.
- The test case should be able to reproduce the issue and verify that it is fixed by running the updated code.
- The corrected code should produce the same output as the original code for all inputs, including `None`.
- The test case should pass after updating the code to fix the 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/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: This test prevents a potential bug where the default filtering behavior is not correctly applied when the 'run' phase is specified.

Key Assertions:

- The function should return the nodeid of the current file when the phase matches.
- The function should return None when the phase does not match.
- The function should handle cases where the phase is neither 'setup', 'teardown', nor 'run'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	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: Prevents a regression where the test would incorrectly report coverage for phases that are not part of the setup configuration.

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: This test prevents a regression where the teardown phase is not properly filtered, potentially leading to false positives in coverage reports.

Key Assertions:

- mapper._extract_nodeid('test_foo.py::test_bar|teardown') == 'test_foo.py::test_bar'
- assert mapper._extract_nodeid('test_foo.py::test_bar|run') is None
- assert mapper._extract_nodeid('test_foo.py::test_bar|setup') 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, 233-234, 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 path is provided.

Why Needed: This test prevents a potential bug where the `extract_nodeid` method incorrectly includes the phase delimiter in the node ID.

Key Assertions:

- The node ID should be returned as-is without any additional characters (e.g., '::').
- The node ID should not include the phase delimiter (e.g., ':').
- The node ID should match the expected path (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	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` 24ms 6

AI ASSESSMENT

Scenario: The test verifies that the _extract_contexts function correctly extracts all contexts for a full logic coverage.

Why Needed: This test prevents regression in coverage calculation by ensuring that all paths are covered, especially when dealing with complex codebases like app.py.

Key Assertions:

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

COVERAGE

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

Why Needed: This test prevents a regression where the coverage mapper fails to extract contexts from files without any test contexts.

Key Assertions:

- mocked_contexts_by_lineno returns an empty dictionary for all files.
- result is an empty dictionary.
- assert result == {} checks if the function returns 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 Extract Node ID Variants for Coverage Mapping.

Why Needed: This test prevents regression in coverage mapping when the target phase is missing.

Key Assertions:

- The function _extract_nodeid should return the correct node ID based on the target phase.
- The function _extract_nodeid should filter out lines that are not under the target phase.
- The function _extract_nodeid should handle cases where there are no lines in the target phase.
- The function _extract_nodeid should raise an assertion error when the target phase is missing.
- The function _extract_nodeid should return None for lines that are not under the target phase and have a 'run' or 'teardown' phase.
- The function _extract_nodeid should handle cases where there are multiple phases with the same name but different names.
- The function _extract_nodeid should raise an assertion error when the target phase is missing and has no lines in 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/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 when no coverage files exist.

Why Needed: This test prevents a potential bug where the function does not handle the case when there are no coverage files.

Key Assertions:

- The function should return None for _load_coverage_data() without any .coverage files.
- There should be exactly one warning with code 'W001' in the warnings list.
- No coverage data should 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/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 that the test_load_coverage_data_read_error function handles errors reading coverage files correctly.

Why Needed: This test prevents a potential regression where the CoverageMapper class fails to handle cases where it encounters an error while loading coverage data from a file.

Key Assertions:

- The function should return None when attempting to load coverage data that is corrupt or cannot be read.
- Any warnings generated by the mapper should contain the message 'Failed to read coverage data'.
- The mapper's warnings should not be empty.
- The function should raise an exception instead of returning None for invalid input.
- The function should handle the case where the coverage file is missing or does not exist.
- Any exceptions raised during the execution of the test should be caught and reported as part of the warning 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/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` 3ms 4

AI ASSESSMENT

Scenario: Test 'test_load_coverage_data_with_parallel_files' verifies that the test suite handles parallel coverage files correctly.

Why Needed: This test prevents a regression where the test fails to detect coverage data from parallel files due to an incorrect mock implementation of `CoverageData`.

Key Assertions:

- The function `_ = mapper._load_coverage_data()` should call `update` on the `CoverageData` instance for each mock `CoverageData` object returned by `mock_data_cls.side_effect`.
- The number of times `update` is called should be greater than or equal to 2.
- The mock instances are correctly mocked with different values.
- The test suite can handle coverage data from parallel files without any issues.
- The test does not fail when the mock implementation of `CoverageData` returns unexpected results.

COVERAGE

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

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	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` 2ms 5

AI ASSESSMENT

Scenario: Verifies that the test handles errors during analysis correctly.

Why Needed: To prevent a regression where an error during analysis would cause missing coverage data.

Key Assertions:

- The mock analysis2 function should be called with an exception as its side effect.
- The mock_data.get_data() method should return the expected mock_data object.
- The entries list should contain no elements after calling map_source_coverage().
- The len(entries) attribute of the entries list should equal 0.
- An error message should be raised when analysis2 fails.

COVERAGE

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

Why Needed: This test prevents regression by ensuring that the coverage analysis is comprehensive and accurate.

Key Assertions:

- The function `mapper.map_source_coverage(mock_cov)` should return exactly one entry with the file path `app.py`.
- The first entry in the returned list should have a total of 3 statements.
- The first entry should have a coverage percentage of 66.67%.
- All statements in the first entry should be covered by the analysis.
- No statements in the first entry should be missed by the analysis.
- The coverage percentage for all statements in the first entry should be greater than or equal to 50%.
- The `covered` attribute of the first entry should be 2 (i.e., both statements are covered).

COVERAGE

src/pytest_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 factory function to ensure it correctly returns a WarningCode.W001_NO_COVERAGE instance with the specified detail.

Why Needed: This test prevents a potential bug where an invalid code is returned from the make_warning function, potentially causing unexpected behavior in downstream code.

Key Assertions:

- The function `make_warning` should return an instance of class WarningCode.W001_NO_COVERAGE with the specified detail.
- The message attribute of the warning object should contain 'No .coverage file found'.
- The detail attribute of the warning object should be set to 'test-detail'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Prevents a potential bug where incorrect warning code values are used, potentially leading to unexpected behavior or errors in the application.

Key Assertions:

- {'message': 'assert WarningCode.W001_NO_COVERAGE.value == "W001"', 'description': 'Correctly asserts that W001_NO_COVERAGE has the correct value'}
- {'message': 'assert WarningCode.W101_LLM_ENABLED.value == "W101"', 'description': 'Correctly asserts that W101_LLM_ENABLED has the correct value'}
- {'message': 'assert WarningCode.W201_OUTPUT_PATH_INVALID.value == "W201"', 'description': 'Correctly asserts that W201_OUTPUT_PATH_INVALID has the correct value'}
- {'message': 'assert WarningCode.W301_INVALID_CONFIG.value == "W301"', 'description': 'Correctly asserts that W301_INVALID_CONFIG has the correct value'}
- {'message': 'assert WarningCode.W401_AGGREGATE_DIR_MISSING.value == "W401"', 'description': 'Correctly asserts that W401_AGGREGATE_DIR_MISSING has 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/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 the Warning.to_dict() method to ensure it returns a dictionary with required keys.

Why Needed: This test prevents a potential issue where the Warning.to_dict() method does not return a dictionary with all required keys, potentially leading to unexpected behavior or errors in downstream code.

Key Assertions:

- The 'code' key should be present and have the correct value.
- The 'message' key should be present and have the correct value.
- The 'detail' key should be present and have the correct value.
- All required keys ('code', 'message', and 'detail') should be included in the dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 makes sure a warning is created with the correct code and message for known code.

Why Needed: Prevents regression in case of unknown code, where the test fails when it's not clear what WarningCode.W101_LLM_ENABLED should be used.

Key Assertions:

- The function make_warning returns a WarningWarning object with the correct WarningCode.W101_LLM_ENABLED.
- The message returned by make_warning is set to WARNING_MESSAGES[WarningCode.W101_LLM_ENABLED].
- The detail of the warning object is None, indicating that no additional information was provided.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Test MakeWarning::test_make_warning_unknown_code verifies that the test uses a fallback message for unknown code when it is not an enum.

Why Needed: This test prevents a regression where the test fails to use a fallback message for unknown code if the WarningCode enum is not allowed.

Key Assertions:

- The function `make_warning(missing_code)` returns a warning message that matches 'Unknown warning.'
- The value of `WARNING_MESSAGES[missing_code]` after restoring the original message is set to `old_message`
- The test assertion checks if the returned warning message matches '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_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 issue where a warning is not generated when an invalid configuration value is provided.

Key Assertions:

- w.code == WarningCode.W301_INVALID_CONFIG
- w.detail == 'Bad value'

COVERAGE

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

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

AI ASSESSMENT

Scenario: Ensures that enum values are correctly defined as strings.

Why Needed: Prevents a potential bug where enum values could be incorrectly typed as integers or other types.

Key Assertions:

- The `value` attribute of each `WarningCode` enum member is an instance of the `str` type.
- Each `WarningCode` value starts with the string '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_dict method of Warning class to ensure it serializes correctly without detail.

Why Needed: This test prevents a potential bug where warnings are not properly serialized to dictionaries, potentially leading to incorrect data being stored or transmitted.

Key Assertions:

- The 'code' key in the dictionary should be set to 'W001'.
- The 'message' key in the dictionary should be set to 'No coverage'.
- The value of 'code' should match the expected string 'W001'.
- The value of 'message' should match the expected string 'No coverage'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/errors.py`	`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 dict method with detailed information.

Why Needed: This test prevents a potential bug where the Warning class's detail field is not properly serialized in its dictionary representation.

Key Assertions:

- The 'detail' field of the Warning object should be present and contain the expected string value.
- The 'code' field of the Warning object should match the expected string value.
- The 'message' field of the Warning object should match the expected string value.
- The 'data' attribute of the Warning object should contain a dictionary with the correct keys and values.
- The 'detail' key in the dictionary should have the correct value.
- The 'code' key in the dictionary should have the correct value.
- The 'message' key in the dictionary should have the correct value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	6 lines (ranges: 70-72, 74-76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 Python files as non-Python files.

Key Assertions:

- The function should return False for files with extensions other than `.py`.
- The function should not return True for files without any Python extension (e.g., `foo.txt`).
- The function should raise a meaningful error or exception when given an invalid file path.
- The function should handle cases where the file is missing a Python extension correctly.
- The function should be able to identify `.pyc` files as valid Python files.
- The function should not incorrectly identify `__init__.py` files as non-Python files.
- The function should raise an error when given a file path that does not exist.

COVERAGE

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

Why Needed: Prevents a potential bug where the function incorrectly returns False for Python files.

Key Assertions:

- The function should return `True` for files with extension `.py`.
- The function should raise an error or behave unexpectedly if it encounters a file without an extension.
- The function should handle files with different casing (e.g., `Foo.py`) correctly.
- The function should not incorrectly identify non-Python files as Python files.
- The function should be able to handle files with multiple extensions (e.g., `.pyc`, `.pyo`).
- The function should raise an error if it encounters a file without a valid extension.
- The function should correctly handle relative paths for .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/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

Scenario: Test 'test_makes_path_relative' verifies that the `make_relative` function correctly makes a path relative to the test directory.

Why Needed: This test prevents regression in cases where the test directory is not in the same location as the test file.

Key Assertions:

- The resulting file path should be 'subdir/file.py' (i.e., relative to the test directory).
- The `parent` argument of `make_relative()` should create a new parent directory if necessary, and `exist_ok=True` should not raise an error.
- The `touch()` method should create the file with the correct permissions.
- The resulting file path should be identical when compared using `os.path.normpath()`.
- The `make_relative()` function should return the expected output string ('subdir/file.py') in all test runs.
- The `make_relative()` function should not raise an error if the input file does not exist (i.e., `exist_ok=True` is used).
- The `make_relative()` function should create a new directory with the correct permissions 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/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: tests/test_fs.py::TestMakeRelative::test_returns_normalized_with_no_base

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

Key Assertions:

- The function `make_relative` should return the original path 'foo/bar' when no base is specified.
- The function `make_relative` should not modify the input path by default.
- The function `make_relative` should preserve the directory structure of the original path.
- The function `make_relative` should handle absolute paths correctly.
- The function `make_relative` should return a normalized path with no base.
- The function `make_relative` should not raise an exception when given invalid input (e.g., empty string).
- The function `make_relative` should have a clear and consistent naming convention.

COVERAGE

src/pytest_llm_report/collector.py	14 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: The 'normalize_path' function should be able to handle paths that are already normalized.

Why Needed: This test prevents a potential bug where the function would incorrectly report an error for already-normalized paths, causing unexpected behavior or incorrect output.

Key Assertions:

- The function should return the original path when given an already-normalized string.
- The function should not throw any errors for already-normalized strings.
- The function should preserve the original directory structure of the input path.
- The function should handle paths with multiple separators correctly (e.g., Windows-style vs. Unix-style).
- The function should not incorrectly report an error for paths that are already normalized (e.g., /foo/bar instead of /foo%2Fbar).

COVERAGE

src/pytest_llm_report/collector.py	14 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

Scenario: Tests the `normalize_path` function with a scenario where forward slashes are used in the path.

Why Needed: Prevents regression when using forward slashes as directory separators.

Key Assertions:

- The function should correctly replace backslashes (`\`) with forward slashes (`/`).
- The resulting string should be identical to the expected output.
- The function should handle paths with multiple consecutive forward slashes correctly.
- The function should not modify the original path.
- The function should raise a `ValueError` when encountering invalid characters in the path.
- The function should support both Unix-style and Windows-style file separators.
- The function should be able to handle paths with leading or trailing whitespace.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: The test verifies that the `normalize_path` function correctly strips trailing slashes from file paths.

Why Needed: This test prevents a potential bug where a file path with a trailing slash is not stripped, potentially leading to unexpected behavior or errors in subsequent operations.

Key Assertions:

- The input string should be modified to have no trailing slash.
- The output string should be the original input string without any trailing slash.
- Any additional slashes in the input string should be removed from the end of the path.
- The function should handle paths with multiple consecutive slashes correctly.
- The function should not modify the input string if it already has no trailing slash.
- The function should raise an error if the input is a directory or a symbolic link instead of a file.
- The function should preserve the original directory structure and only remove the 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: Test verifies whether a path matches custom exclusion patterns.

Why Needed: This test prevents the inclusion of paths that match custom patterns ('test*') in the build process.

Key Assertions:

- The function `should_skip_path` is called with the correct arguments.
- The expected result for 'tests/conftest.py' matches the actual result.
- The expected result for 'src/module.py' does not match the actual result.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 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: tests/test_fs.py::TestShouldSkipPath::test_normal_path

Why Needed: Prevents skipping of normal file system paths.

Key Assertions:

- The function `should_skip_path` should return `True` for a normal path.
- The function `should_skip_path` should not be called with a normal path.
- The test should verify that the assertion is correctly set to `False` for normal paths.
- The test should verify that the function does not raise an exception when given a normal path.
- The test should verify that the function does not return `None` when given a normal path.
- The test should verify that the function returns `True` when given a non-normal path.
- The test should verify that the function is called with the correct arguments for normal 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	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 function `should_skip_path` correctly identifies `.git` directories.

Why Needed: This test prevents a potential bug where the function incorrectly skips non-existent `.git` directories, causing unexpected behavior in downstream code.

Key Assertions:

- assert should_skip_path('.git/objects/foo') is True
- assert should_skip_path('non_existent_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: The test verifies that the `should_skip_path` function correctly identifies and skips `__pycache__` directories.

Why Needed: This test prevents a potential issue where the test incorrectly includes `__pycache__` directories in the skipped paths.

Key Assertions:

- assert should_skip_path('foo/__pycache__/bar.pyc') is True
- assert should_skip_path('non-existent-path/__pycache__/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	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 `should_skip_path` function incorrectly considers venv directories as Python site packages.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 false positives for non-python files, ensuring accurate coverage of Python files.

Key Assertions:

- assert is_python_file('module.txt') is False
- assert is_python_file('module.pyc') is False
- assert is_python_file('module') 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_coverage.py::TestIsPythonFile::test_is_python_file_true 1ms 3

AI ASSESSMENT

Scenario: Testing whether a module is a Python file.

Why Needed: Prevents a False positive where non-Python files are incorrectly identified as such.

Key Assertions:

- The function `is_python_file` correctly identifies .py files and their subdirectories.
- It handles relative paths to .py files correctly.
- It ignores Python file extensions other than .py.
- It does not incorrectly identify non-Python files with a similar extension (e.g., .java, .c).
- It handles absolute paths to .py files correctly.
- It correctly identifies .py files in subdirectories of the current working directory.
- It ignores Python file names other than those starting with a letter or underscore.
- It does not incorrectly identify non-Python files with a similar name (e.g., `module.js`, `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	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 that is not under the base directory.

Why Needed: Prevents regression where make_relative returns an incorrect absolute path when the input path is not relative to the base.

Key Assertions:

- The result should contain 'project1' as the first component of the normalized absolute path.
- The result should contain 'file.py' as the second component of the normalized absolute path.
- The result should be an absolute path.

COVERAGE

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

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_coverage.py::TestMakeRelative::test_make_relative_with_none_base` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `make_relative` function correctly handles cases where the base is None.

Why Needed: This test prevents a potential bug where the relative path would be normalized to include the base directory, potentially leading to unexpected behavior or errors.

Key Assertions:

- The result of `make_relative('path/to/file.py', None)` should be 'path/to/file.py'.
- The absolute path returned by `make_relative` is not affected by the presence of a base directory.
- The relative path is correctly normalized to exclude the base directory.
- The test does not fail when the base is None, ensuring correctness.
- The test does not pass without asserting that the result matches the expected value.
- The function behaves as expected when given an empty string as the base directory.
- The function handles cases where the input path has a relative extension 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	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: The 'test_normalize_path_backslashes' test verifies that the `normalize_path` function correctly converts backslashes in file paths to forward slashes.

Why Needed: This test prevents a potential issue where Windows-style file paths with backslashes are not properly normalized, leading to unexpected behavior or errors when comparing against expected results.

Key Assertions:

- The normalized path should have the same contents as the original unnormalized path.
- The normalized path should start with a forward slash (/).
- Backslashes in the original path should be replaced with forward slashes in the normalized path.
- No leading or trailing backslashes are present in the normalized path.
- The normalized path does not contain any double backslashes (\).

COVERAGE

src/pytest_llm_report/collector.py	14 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

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_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 issues where paths with trailing slashes may not be correctly normalized to their base form.

Key Assertions:

- The function `normalize_path` removes any trailing slash from the input path.
- If the input path has a trailing slash, it is removed and returned as the normalized path.
- The normalization of a path with a trailing slash does not affect its semantic meaning or functionality.
- Paths with multiple consecutive slashes are correctly handled by `normalize_path`.
- The function handles paths with leading slashes correctly.
- If the input path has only one slash, it is preserved as-is and returned as the normalized path.
- The normalization of a path with a trailing slash does not introduce any new errors or inconsistencies.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Test 'test_should_not_skip_regular_path' verifies whether regular paths are not skipped.

Why Needed: This test prevents skipping of regular paths, ensuring that all files are covered under the test coverage.

Key Assertions:

- should_skip_path('src/module.py') is False
- should_skip_path('tests/test_foo.py') 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	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 potential issue where the test might incorrectly skip important Git hooks or directories.

Key Assertions:

- The function should return `True` for the `.git/hooks/pre-commit` file.
- The function should not return `False` for other `.git` directory files (e.g., `.gitignore`, `.gitattributes`).
- The function should handle any potential exceptions that may occur when trying to access or read the `.git` directory.
- The function should correctly identify and skip Git hooks, while still allowing other important directories to be processed.
- The function's behavior should not change if the test is run multiple times with different input files.
- The function should provide a clear indication that it has successfully skipped the `.git` directory when done.

COVERAGE

`src/pytest_llm_report/collector.py`	14 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 skips paths starting with a 'skip_dir' name.

Why Needed: This test prevents regression in the 'should_skip_path' function, which may cause it to incorrectly skip certain paths.

Key Assertions:

- If the path starts with 'venv', the function should return True indicating that it should be skipped.
- If the path starts with '.venv', the function should also return True indicating that it should be skipped.
- The function should handle cases where the path is a subdirectory of the skip_dir name 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_pycache` 1ms 3

AI ASSESSMENT

Scenario: Verify the correctness of `should_skip_path` function.

Why Needed: Prevents a potential bug where `__pycache__` directories are incorrectly included in coverage reports.

Key Assertions:

- The test verifies that `should_skip_path` returns True for paths within `src/__pycache__/`.
- It checks if the function correctly identifies and excludes `__pycache__` directories from coverage reporting.
- The test ensures that the function handles cases where a file is present in `__pycache__` but not executable or modified.
- It verifies that the function does not include paths with non-executable files (e.g., `.gitignore`, etc.) within `__pycache__`.
- The test checks if the function correctly excludes directories with symbolic links to other directories within `__pycache__`.
- It ensures that the function handles cases where a file is present in `__pycache__` but not a regular file (e.g., a directory, etc.).
- The test verifies that the function does not include paths with non-executable files or symbolic links to other directories within `__pycache__`.

COVERAGE

src/pytest_llm_report/collector.py	14 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 by the test.

Why Needed: Prevent a regression where site-packages directories are not correctly identified as to be skipped.

Key Assertions:

- The path '/usr/lib/python3.12/site-packages/pkg/mod.py' should be considered a site-packages directory.
- The 'site-packages' directory is expected to be skipped by the test.
- The test should fail when attempting to access the 'pkg/mod.py' file within the 'site-packages' directory.
- The 'site-packages' directory should not contain any files that require direct access (e.g., executables, configuration files).
- The test should correctly identify and skip all site-packages directories in the given path.
- The 'site-packages' directory should be skipped when the test is run with a specific set of dependencies or environment variables.
- The test should handle any potential exceptions that may arise while attempting to access the 'pkg/mod.py' file within the 'site-packages' 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	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 correctly skipped.

Why Needed: This test prevents a potential issue where venv directories are not properly excluded from coverage reports.

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: Verify the test 'test_should_skip_with_exclude_patterns' checks if custom exclude patterns prevent skipping of paths with '*'secret*' in them.

Why Needed: This test prevents a regression where custom exclude patterns are not correctly preventing path skipping when they contain '*'secret*' keywords.

Key Assertions:

- The function should return True for the given path and exclude pattern.
- The function should return False for the given path and exclude pattern.
- Custom exclude patterns should prevent path skipping with '*'secret*' in them.
- Custom exclude patterns should not prevent path skipping without '*'secret*' keywords.
- The test should be able to correctly handle cases where the excluded file is not found or does not exist.
- The function should raise an exception when the excluded pattern matches a non-existent file.
- The function should return False for paths that do not contain any exclude patterns.
- Custom exclude patterns should work as expected even with multiple '*'secret*' in them.

COVERAGE

src/pytest_llm_report/collector.py	14 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 clears request and token usage records after a past request.

Why Needed: This test prevents the gemini rate limiter from incorrectly storing requests and tokens in the database for an extended period, potentially leading to performance issues or security vulnerabilities.

Key Assertions:

- The length of _request_times should be 0 after pruning.
- The length of _token_usage should be 0 after pruning.
- _request_times.append(time.monotonic() - 61) should not store a request in the database for more than 1 minute.
- _token_usage.append((time.monotonic() - 61, 100)) should not store a token usage record in the database for more than 1 minute.
- limiter._prune(time.monotonic()) should clear _request_times and _token_usage records after pruning.
- assert len(limiter._request_times) == 0 should be true after pruning.
- assert len(limiter._token_usage) == 0 should be true after pruning.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	11 lines (ranges: 39-42, 81-85, 87-88)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

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

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents requests from exceeding the specified limit in a realistic scenario.

Why Needed: This test prevents a potential regression where the rate limiter allows too many requests to reach the RPM limit, potentially causing performance issues.

Key Assertions:

- The `next_available_in` method should return a non-negative value greater than 0 and less than or equal to 60.0 seconds.
- The `wait` assertion should be true after calling `next_available_in` with the specified limit (1 requests per minute).
- The rate limiter's `record_request` method should not have been called before checking if there are any available requests.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	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 the TPM (Tokens Per Minute) limit is correctly enforced when tokens are recorded at a rate higher than the configured limit.

Why Needed: This test prevents a potential bug where the TPM limit is not respected if tokens are recorded too quickly, potentially leading to token starvation or other issues.

Key Assertions:

- The next_available_in method returns a value greater than 0 when there are available tokens in the limiter.
- The length of _token_usage is equal to 2 after recording tokens at rate higher than the configured limit.
- The number of tokens recorded increases by 10 and then decreases back to 90, 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/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: Verify that the `wait_for_slot` method sleeps when a request is recorded.

Why Needed: This test prevents a potential issue where the rate limiter does not sleep when a new request is recorded, potentially leading to unexpected behavior in production.

Key Assertions:

- limiter.wait_for_slot(1) was called with mock_sleep
- the `mock_sleep` object has a `called` attribute indicating that it was called once
- the `time.sleep` function was not called within the specified time frame (1 minute)
- no other requests were recorded during the test duration

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/gemini.py`	`31 lines (ranges: 39-42, 45-46, 48, 52-54, 58-59, 61-63, 73, 76-78, 81-82, 84, 87-88, 92-93, 95, 97, 100-102)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 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 bug where the rate limiter does not record tokens for a long time, potentially leading to incorrect usage statistics.

Key Assertions:

- The function `_GeminiRateLimitConfig` is called with `tokens_per_minute=100`.
- The function `_GeminiRateLimiter` is called with no arguments.
- The limiter's `_token_usage` list is empty after calling `record_tokens(0)`.
- The length of the `_token_usage` list is 0.
- _GeminiRateLimitConfig returns a valid instance of _GeminiRateLimitConfig.
- _GeminiRateLimiter does not raise an exception when called with no arguments.
- The limiter's `_token_usage` list contains only one element (0 tokens).
- The length of the `_token_usage` list is 1 after calling `record_tokens(0)`.

COVERAGE

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

PASSED `tests/test_gemini_coverage_v2.py::test_gemini_limiter_requests_per_day_exhaustion` 2ms 3

AI ASSESSMENT

Scenario: Verify that the test ensures a daily limit is exceeded when requests are made too frequently.

Why Needed: This test prevents a rate limiting error where the daily limit is not reached even though many requests are being made.

Key Assertions:

- The limiter raises an exception with the message 'requests_per_day' when the number of requests exceeds the daily limit.
- The limiter waits for a slot to become available after exceeding the daily limit.
- The limiter records a request before attempting to wait for a slot.
- The limiter checks if the number of requests per day is exceeded and raises an exception accordingly.
- The limiter attempts to wait for a slot when the daily limit has been reached, resulting in an error.
- The limiter records multiple requests within a short period, exceeding the daily limit.
- The limiter waits for a long time after exceeding the daily limit before attempting to wait again.
- The limiter checks if the number of requests per day is exceeded and raises an exception accordingly.

COVERAGE

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

Why Needed: Prevents a potential rate limiter timeout error due to excessive tokens used in a short period.

Key Assertions:

- assert wait > 0
- assert 'tokens_used' in request_tokens
- request_tokens is not empty
- tokens_used + request_tokens <= limit
- limiter._seconds_until_tpm_available(now, 5) == 0

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 576ms 6

AI ASSESSMENT

Scenario: The test verifies that the RPM rate limit cooldown handling works correctly when a call to `_GeminiRateLimitExceeded` is made with an insufficient number of retries.

Why Needed: This test prevents regression where the RPM rate limit cooldown is not set properly, potentially leading to unexpected behavior or errors in subsequent calls.

Key Assertions:

- The `models/gemini-pro` model should be present in the `_cooldowns` dictionary.
- The value of `_cooldowns['models/gemini-pro']` should be greater than 1000.0 seconds (1 minute) to reflect a valid cooldown period.
- The RPM rate limit cooldown handling should not set the `models/gemini-pro` model in the `_cooldowns` dictionary for the first call, allowing it to fail with an error message.
- After retrying the failed call, the cooldown period should be reset to 1000.0 seconds (1 minute) and the RPM rate limit should be re-enabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: Verify that the GeminiProvider correctly annotates a model after rate limit retry attempts

Why Needed: This test prevents regression in case of rate limit retries, ensuring that the annotation process is consistent and reliable.

Key Assertions:

- The annotation result should match the expected scenario 'Recovered Scenario'
- Mock _parse_response was called exactly twice with the correct arguments
- The mock post call count matches the actual number of calls made to the provider
- The provider's internal annotation process returns a valid annotation object with the correct scenario and error status

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 correctly annotates a success scenario with the expected key assertions.

Why Needed: This test prevents regression in case of an incorrect annotation format, where _parse_response might expect a specific structure instead of the actual response from _call_gemini.

Key Assertions:

- The annotation object should have a 'scenario' attribute equal to 'Success Scenario'.
- The annotation object should not have an 'error' attribute. If it does, this test will fail and provide a meaningful error message.
- The annotation object's 'scenario' attribute should be set correctly from the response of _call_gemini.
- The annotation object's 'error' attribute should be None if no error is present in the response of _parse_response.
- The annotation object's 'text' attribute should match the expected text for a success scenario.
- The annotation object's 'tokens' attribute should not contain any tokens if there are none to annotate.
- The annotation object's 'source' attribute should be set correctly from the test result.
- The annotation object's 'totalTokenCount' attribute should match the total token count in the response of _parse_response.
- The annotation object's 'candidates' attribute should contain a list with one item matching the expected scenario.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: Test availability of the GeminiProvider when no API token is provided

Why Needed: This test prevents a potential bug where the provider attempts to use an invalid API token.

Key Assertions:

- provider._check_availability() should return False when no GEMINI_API_TOKEN environment variable is set
- provider._check_availability() should raise an AssertionError when no GEMINI_API_TOKEN environment variable 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/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

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/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 block subsequent requests after the first two are processed.

Why Needed: This test prevents a potential issue where the rate limiter blocks subsequent requests, potentially causing unexpected behavior or errors.

Key Assertions:

- The next_available_in method should return 0.0 for the first two requests.
- The next_available_in method should return 0.0 after recording the third request.
- wait should be greater than 0 and less than or equal to 60.0 for subsequent requests.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	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: The 'different_config' test verifies that two different configuration providers result in distinct hash values.

Why Needed: This test prevents a potential issue where the same configuration provider could produce identical hashes due to caching or other internal implementation details.

Key Assertions:

- config1 and config2 have different provider strings
- compute_config_hash(config1) is not equal to compute_config_hash(config2)
- compute_config_hash(config1) should be a unique value for each configuration

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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: Verifies the length of the computed hash is 16 characters.

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

Key Assertions:

- The length of the computed hash should be exactly 16 characters.
- The computed hash should not exceed 15 characters to prevent potential issues.
- The computed hash should start and end with a non-zero value (0x...).
- No leading zeros are allowed in the computed hash.
- No trailing zeros are allowed in the computed hash.
- All characters in the computed hash must be hexadecimal digits or 0-9.
- No special characters, whitespace, or other non-hexadecimal characters should be present in the computed hash.
- The computed hash should not contain any repeated values (no duplicates).

COVERAGE

src/pytest_llm_report/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 the consistency of a file's hash with its content hash when computed using `compute_file_sha256()`.

Why Needed: This test prevents regression where the file's hash does not match its content hash after re-computing it using `compute_file_sha256()`.

Key Assertions:

- The file's hash should be equal to its content hash when computed using `compute_file_sha256()`.
- The file's hash should be equal to its content hash when the file is read from disk and re-computed using `compute_file_sha256()`.
- The file's hash should not change after re-computing it using `compute_file_sha256()` if the original file was already hashed correctly.
- The file's hash should match the expected value returned by `compute_file_sha256()` when computed directly from the file contents.
- The file's hash should be equal to its content hash when the file is re-computed using a different hashing algorithm or library.
- The file's hash should not change after re-computing it using `compute_file_sha256()` if the original file was already hashed correctly and the input data has not changed.

COVERAGE

src/pytest_llm_report/collector.py	14 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 potential issues where the computed hash does not match the expected output due to incorrect or missing file contents.

Key Assertions:

- The length of the computed hash should be exactly 64 bytes.
- The computed hash should contain all characters from the input string 'hello world'.
- No whitespace characters should be present in the computed hash.
- No duplicate bytes should be present in the computed hash.
- All hexadecimal digits should be present in the computed hash.
- The computed hash should not be empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Verifies that the computed HMAC has the correct length for a given input.

Why Needed: Prevents a potential issue where an incorrect or truncated HMAC is generated.

Key Assertions:

- The computed HMAC should have a length of 64 bytes.
- The HMAC should be generated using the provided secret key.
- The HMAC should not be truncated or shortened 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	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

Scenario: Test 'Same content should produce same hash' verifies that the output of computing SHA-256 on a piece of text is consistent.

Why Needed: This test prevents regressions where the SHA-256 hash of the same input changes over time due to external factors such as network latency or system updates.

Key Assertions:

- The function `compute_sha256(b'...')` returns the same output for two identical inputs.
- The function `compute_sha256(b'...')` raises an AssertionError when given different inputs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: The hash function is expected to produce a SHA-256 hash of the input string "test".

Why Needed: This test prevents a potential issue where the hash length might not be as expected due to incorrect implementation or configuration of the hashing algorithm.

Key Assertions:

- The length of the resulting hash should be exactly 64 characters (0x400).
- The hash should contain all 256 possible hexadecimal digits (0-9, a-f, A-F).
- No leading zeros are allowed in the hash. The first character should not be zero.
- The hash is not empty and does not start with a zero.
- All characters in the input string are present in the output hash.
- No duplicate characters are present in the output hash.
- The hash is a valid SHA-256 hash (e.g., 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/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: Verifies that the `get_dependency_snapshot` function includes the 'pytest' package.

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

Key Assertions:

- The 'pytest' package should be present in the dependency snapshot.
- The 'pytest' package should be listed as an item in the dependency snapshot.
- The presence of 'pytest' in the dependency snapshot should be verified using `assert 'pytest' in snapshot`.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 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` 83ms 3

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the function might not return a dictionary or may return incorrect data.

Key Assertions:

- snapshot is an instance of dict
- snapshot has no attributes other than __dict__
- snapshot does not contain any packages
- snapshot contains only dependencies without their versions
- snapshot contains all required keys (name, version, etc.) in the correct order

COVERAGE

src/pytest_llm_report/collector.py	14 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

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/util/hashing.py	5 lines (ranges: 73, 76-77, 80-81)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the function would return an HMAC key if a non-existent key file was used, potentially leading to unexpected behavior or errors in downstream applications.

Key Assertions:

- The Config object should be created with a valid key file path.
- The load_hmac_key function should raise a ValueError when given an invalid key file path.
- The assert statement should return None for the HMAC key.
- The Config object's hmac_key attribute should not contain the non-existent key file path.
- The load_hmac_key function should not return an HMAC key if the provided key file does not exist.
- A ValueError should be raised when trying to create a Config object with a non-existent key file.
- The assert statement should fail for the HMAC 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	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: Test that the function returns None when no key file is specified.

Why Needed: This test prevents a potential bug where the function does not return an error message or exception when no key file is provided.

Key Assertions:

- The function should throw an AssertionError with a meaningful message if no key file is configured.
- The function should raise a ConfigError indicating that no key file was specified.
- The function should return None as expected when no key file is loaded.
- The function should not load any HMAC keys from the configuration.
- The function should handle invalid or missing key files gracefully without crashing.

COVERAGE

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

Why Needed: Prevents regression in aggregation settings when no aggregation policy is specified.

Key Assertions:

- The `aggregate_dir` attribute of the test configuration is set to None.
- The `aggregate_policy` attribute of the test configuration is set to 'latest'.
- The `aggregate_include_history` attribute of the test configuration is set to False.

COVERAGE

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

Why Needed: This test prevents a regression where the default capture failed output was incorrectly set to True.

Key Assertions:

- The `capture_failed_output` attribute of the configuration object is set to `False`.
- The value of `capture_failed_output` in the configuration object matches its expected default value.
- If the `capture_failed_output` attribute is not set, an exception is raised with a meaningful error message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: Verify that the context mode is set to 'minimal' by default.

Why Needed: This test prevents a potential regression where the context mode might be set to 'default' instead of 'minimal'.

Key Assertions:

- config.llm_context_mode == 'minimal'
- config.llm_context_mode != 'default'
- config.llm_context_mode == '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_llm_not_enabled_by_default` 1ms 3

AI ASSESSMENT

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

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

Key Assertions:

- config.is_llm_enabled() == False
- config.get_llm_enabled_state() != 'enabled'
- config.get_llm_enabled_value() == None
- get_default_config().llm_enabled() == False
- get_default_config().is_llm_enabled() == False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	4 lines (ranges: 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 no other configuration options are provided.

Why Needed: This test prevents a regression where the default behavior of omitting tests from coverage is not enforced.

Key Assertions:

- The `config.omit_tests_from_coverage` attribute is set to `True`.
- The `get_default_config()` function returns an instance with `omit_tests_from_coverage` set to `True`.
- The `assert` statement checks that the value of `config.omit_tests_from_coverage` matches the expected result.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/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: Verify that the provider defaults to 'none' when no provider is specified.

Why Needed: Prevents a potential bug where the default provider is set to 'default', potentially leading to unexpected behavior in downstream code.

Key Assertions:

- The `config.provider` attribute of the test configuration object should be equal to 'none'.
- If no provider is specified, the `provider` value should be 'none'.
- The `get_default_config()` function returns a test configuration object with a `provider` attribute set to 'none'.

COVERAGE

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

Scenario: Verify that secret files are excluded by default.

Why Needed: This test prevents a potential bug where the default configuration does not exclude secret files, potentially leading to sensitive data exposure.

Key Assertions:

- The 'secret' keyword is present in the list of excludes.
- The '.env' file is present in the list of excludes.
- All secret files are excluded from the llm context.
- The 'secret' keyword is not present in any exclude list.
- The '.env' file is not present in any exclude list.
- No other sensitive files are excluded by default.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 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: Test that the output of `test_deterministic_output` function is deterministic and correctly sorted by nodeid.

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

Key Assertions:

- nodeids should be in ascending order (sorted)
- all tests have the same nodeid
- the nodeids list does not contain duplicates
- each test has a unique nodeid
- no duplicate nodeids are present
- tests are ordered by their nodeid

COVERAGE

src/pytest_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` 7ms 5

AI ASSESSMENT

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

Why Needed: This test prevents regression where the test suite is empty and still produces a valid report.

Key Assertions:

- The total count of tests in the report should be zero.
- All test counts in the report should match the number of tests run.
- The 'total' key in the summary section should contain zero.
- Each test count in the report should be an integer.
- There should be no missing or invalid data in the report.
- The report should not contain any empty lines or comments.
- All test counts should match the number of tests run, including skipped and failed tests.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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` 37ms 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 file 'report.html' exists at the specified path.
- is present in the contents of 'report.html'.
- The string 'test_pass' is found within the contents of 'report.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/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` 62ms 7

AI ASSESSMENT

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

Why Needed: This test prevents regression where the pipeline fails to generate a JSON report even when all tests pass or fail.

Key Assertions:

- The 'report.json' file exists in the expected location.
- The schema version of the report is correct (1.0).
- The total number of tests passed is 3.
- The number of tests that failed is 1.
- The number of tests that were skipped is 1.
- All test nodes have a unique 'nodeid'.

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 object has required fields.

Why Needed: Prevents a potential bug where the report root is missing required fields.

Key Assertions:

- The 'schema_version' field should be present in the report data.
- The 'run_meta' field should be present in the report data.
- The 'summary' field should be present in the report data.
- The 'tests' field should be present in the report 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 'RunMeta has aggregation fields' verifies that the test runs meta contains aggregation fields.

Why Needed: This test prevents regression where 'is_aggregated' and 'run_count' are missing from RunMeta's to_dict() method.

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 'test_run_meta_has_status_fields' verifies that RunMeta has run status fields.

Why Needed: This test prevents regression where the schema compatibility is not enforced and some meta fields are missing.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 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: Verify that the schema version is defined and matches a semver-like format.

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

Key Assertions:

- The `SCHEMA_VERSION` variable should be set to a string value.
- The `SCHEMA_VERSION` variable should contain at least one dot (`.`) character.
- The `SCHEMA_VERSION` variable should be in the correct semver-like format (e.g., '1.2.3'),

COVERAGE

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

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

AI ASSESSMENT

Scenario: The `TestSchemaCompatibility` class is tested to ensure that the generated `TestCaseResult` object has all required fields.

Why Needed: This test prevents a potential bug where the `TestCaseResult` object may be missing one or more required fields, potentially causing issues with data validation and processing.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`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 when all retries are exhausted, the LLMTokenRefreshRetry function returns an annotation with an error.

Why Needed: Prevents regression where the LLMTokenRefreshRetry function fails to return a valid annotation even after exhausting all retries.

Key Assertions:

- The result of calling `provider.annotate()` is not None.
- The `error` attribute of the result is not None.
- The `test_source` and `context_files` are empty.
- The `completion` function raised an exception when called with mock arguments.
- The annotation returned by `provider.annotate()` contains a valid error message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 the LLM may be forced to refresh its token without user input.

Key Assertions:

- The test verifies that the annotation returned by `provider.annotate` contains an error.
- The test verifies that the error is not a 401 status code.
- The test verifies that the error message is not 'Internal server error'.
- The test verifies that the error message is not 'Token refresh failed'.
- The test verifies that the annotation does not contain any additional information related to token refresh.
- The test verifies that the annotation has an `error` key with a non-401 status code.
- The test verifies that the `error` value is an instance of the `Exception` class.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: This test prevents regression where the LLM token refresh fails due to a transient error, causing it to fail subsequent attempts.

Key Assertions:

- The API call is mocked to fail twice before succeeding.
- The mock completion function raises an exception within the retry limit.
- The annotation result does not contain any errors.
- The scenario of the annotation result matches the expected test scenario.
- The number of retries is sufficient for the transient error to be handled.
- The LLM token refresh successfully completes after the 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` 6.40s 7

AI ASSESSMENT

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

Why Needed: To ensure that the token refresh mechanism works correctly when encountering a 401 status code.

Key Assertions:

- The test verifies that the API call is retried after a token refresh attempt with a 401 status code.
- The test ensures that the correct error message is raised for authentication failure.
- The test verifies that the token refresh command is executed correctly with the new token.
- The test checks that the API call is retried at least twice (or not None) after a successful token refresh attempt.
- The test verifies that the annotation result contains the expected key assertions about the API call behavior.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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 `get_provider` function incorrectly returns a different provider type (e.g., 'diamond') when the `provider` parameter is set to 'gemini'.

Key Assertions:

- The `provider` attribute of the returned instance should be an instance of `GeminiProvider`.
- The `__class__.__name__` attribute of the returned instance should match `'GeminiProvider'`.
- The `get_provider` function is called with a valid `Config` object and a correct provider type.
- The `provider` parameter passed to `get_provider` is set to 'gemini'.
- The `config.model` attribute of the `Config` object is set to 'gemini-1.5-flash'.

COVERAGE

src/pytest_llm_report/collector.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: Verify that the test LitEllm returns a LiteLLMProvider instance.

Why Needed: This test prevents regression where the LitEllm model is not correctly identified as a LiteLLMProvider.

Key Assertions:

- The provider attribute of the returned provider object should be 'litellm'.
- The class name of the returned provider object should be 'LiteLLMProvider'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	9 lines (ranges: 52-53, 245, 247, 249, 252, 257-258, 260)
src/pytest_llm_report/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 when 'None' is specified as the provider, it returns a NoopProvider instance.

Why Needed: This test prevents regression where None is passed to the LLM provider.

Key Assertions:

- provider should be set to 'none'
- provider should return an instance of NoopProvider
- NoopProvider should not be created with a valid configuration

COVERAGE

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

Why Needed: This test prevents a potential bug where the correct provider is not retrieved due to an import error if httpx is missing.

Key Assertions:

- provider.__class__.__name__ should be equal to 'OllamaProvider'.
- provider should have the correct type (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 trying to get 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 `Config` class's constructor should be able to create a Config object with an unknown provider.
- The `get_provider` function should not be able to handle an unknown provider without raising a ValueError.
- An AssertionError should be raised when trying to get a provider with an unknown provider.
- The test should fail if the `get_provider` function is called with an unknown provider.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 245, 247, 249, 252, 257, 262, 267)
src/pytest_llm_report/options.py	2 lines (ranges: 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: Test that NoopProvider implements LlmProvider contract.

Why Needed: Prevents regression in the LlmProvider implementation.

Key Assertions:

- The provider should have required methods: annotate, is_available, get_model_name, and config.
- The provider should not be able to provide any LLMs without a valid configuration.
- NoopProvider should not implement all methods of LlmProvider contract.

COVERAGE

src/pytest_llm_report/collector.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 annotation when no node is provided.

Why Needed: This test prevents a regression where the annotate method does not return any annotation for nodes without a scenario.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 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: Verify that the NoopProvider returns an empty string when the model name is not provided.

Why Needed: Prevents a bug where the model name is missing and the provider returns a non-empty string.

Key Assertions:

- assert provider.get_model_name() == ''
- assert provider.get_model_name() != 'noop'
- assert provider.get_model_name().startswith('noop')
- assert not provider.get_model_name().endswith('noop')
- assert len(provider.get_model_name()) == 0

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 66)
src/pytest_llm_report/options.py	2 lines (ranges: 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 always available.

Why Needed: This test prevents a potential bug where the provider might not be available due to an issue with the configuration or initialization.

Key Assertions:

- provider.is_available() should return True
- provider.config is not None
- provider.config is not empty
- provider.config is not None and config is not empty
- provider.config is a valid Config object
- provider.config has the required attributes (e.g., model, tokenizer)
- provider.config has a non-empty model and tokenizer

COVERAGE

src/pytest_llm_report/collector.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: Test annotation summary is printed when annotations run.

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

Key Assertions:

- The function `get_provider` of `llm.annotator` returns a `FakeProvider` instance with the correct scenario.
- The `TestCaseResult` object has an attribute `outcome` set to 'passed'.
- The `pytest_llm_report.llm.annotator.get_provider` function is called with the correct configuration.
- 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 LLM annotator to report progress via callback.

Why Needed: This test prevents regression where the LLM annotator does not report progress.

Key Assertions:

- The test verifies that the LLM annotation reports progress for all tests.
- The test expects the first message to contain 'Starting LLM annotations for 1 test(s)'
- The test checks if the second message contains the expected test case ID.
- The test asserts that the messages list is not empty after calling `append` on it.
- The test verifies that the progress callback returns a non-empty list of messages.
- The test expects the last message to contain 'tests/test_progress.py::test_case'.
- The test checks if the progress callback appends the correct number of messages to the list.

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: tests/test_llm_annotator.py::test_annotate_tests_respects_opt_out_and_limit verifies that LLM annotations do not include opt-out tests and respect the max number of tests.

Why Needed: This test prevents regression where LLM annotations incorrectly include opt-out tests or exceed the maximum allowed number of tests.

Key Assertions:

- The `llm_opt_out` attribute is set to `True` for all test cases.
- Only one LLM annotation is created (i.e., no more than `llm_max_tests=1`) for each test case.
- No opt-out tests are included in the LLM annotations.
- All test cases have an LLM annotation, and there are no duplicate annotations.
- The maximum number of tests is respected (i.e., no more than `llm_max_tests=1`)
- The `provider.calls` list contains only one call to `get_provider` with the correct configuration.
- Each test case has an LLM annotation, and there are no missing annotations.
- No duplicate LLM annotations are created for different test cases.

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 annotations do not exceed the requests-per-minute rate limit.

Why Needed: This test prevents a potential regression where the LLM annotator exceeds the rate limit and fails to report all tests.

Key Assertions:

- The provider's calls should match the expected list of node IDs.
- The sleep function should be called twice with times equal to 2.0 seconds.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-173, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 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 the annotation function skips unavailable providers with a clear message.

Why Needed: This test prevents regression where the annotation function does not skip unavailable providers correctly.

Key Assertions:

- The `is_available` method of the provider returns False for an unavailable provider.
- The `get_provider` function from pytest_llm_report.llm.annotator sets the `provider` attribute to the provided instance.
- The annotation function calls `get_provider` with a valid configuration and then checks if the provider is available.
- If the provider is not available, the annotation function should skip annotating the tests.
- A message indicating that the provider is unavailable should be printed to the console.
- The test should fail when an unavailable provider is used.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 2ms 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 schema requires both 'scenario' and 'why_needed' fields.

Why Needed: This test prevents regression by ensuring that the schema has a minimum of two required fields: 'scenario' and 'why_needed'.

Key Assertions:

- required = ['scenario', 'why_needed']
- scenario in required
- 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: The test verifies that the `AnnotationSchema` can parse a dictionary containing required fields.

Why Needed: This test prevents potential bugs where the `AnnotationSchema` might not be able to correctly extract or validate the necessary information from a malformed input.

Key Assertions:

- checks password
- checks username
- the length of `schema.key_assertions` is equal to 2

COVERAGE

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

Scenario: Verifies that the AnnotationSchema class correctly handles an empty input.

Why Needed: Prevents a potential bug where the schema is not properly initialized with an empty dictionary.

Key Assertions:

- schema.scenario should be an empty string
- schema.why_needed should indicate that the schema needs to be initialized with 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/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: The test verifies that the AnnotationSchema from_dict function correctly handles a partial input scenario.

Why Needed: This test prevents regression where the AnnotationSchema may incorrectly infer the 'scenario' field when provided with incomplete data.

Key Assertions:

- assert schema.scenario == 'Partial only'
- assert schema.why_needed == ''
- schema.scenario should be equal to 'Partial only'
- assert isinstance(schema.why_needed, str)
- schema.why_needed should be an empty string
- schema.scenario is a string
- schema.why_needed is also a string

COVERAGE

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

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

Key Assertions:

- The 'scenario' field should be present in the schema.
- The 'why_needed' field should be present in the schema.
- The 'key_assertions' field should be present in the schema.

COVERAGE

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

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

AI ASSESSMENT

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

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

Key Assertions:

- assertion 1: The 'scenario' field in the resulting dictionary matches the expected value.
- assertion 2: The 'why_needed' field in the resulting dictionary matches the expected value.
- assertion 3: The 'key_assertions' field in the resulting dictionary is present and contains the expected values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	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: Test that the Factory returns a NoopProvider for 'none' provider.

Why Needed: This test prevents a potential bug where the Factory returns a NonOpProvider instead of a NoopProvider when the provider is set to 'none'.

Key Assertions:

- The function `get_provider(config)` should return an instance of `NoopProvider` for the given configuration.
- The `provider` attribute of the returned `NoopProvider` should be equal to `'none'`.
- The `provider` attribute of the returned `NoopProvider` should not be a string like 'some_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 test verifies that a NoopProvider instance is correctly identified as an LlmProvider.

Why Needed: This test prevents the regression where a NoopProvider is incorrectly identified as an LlmProvider.

Key Assertions:

- provider is an instance of LlmProvider
- provider is not None
- provider does not inherit from LlmProvider
- provider has a different configuration than LlmProvider
- provider has the correct methods and attributes for its class
- provider's type is correctly determined by its class

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 the provided function does not have any dependencies.

Why Needed: This test prevents a potential bug where the NoopProvider returns an incorrect annotation for functions with no dependencies, potentially leading to false positives or negatives in the report.

Key Assertions:

- The annotation returned by the NoopProvider is empty.
- The annotation returned by the NoopProvider does not contain any information about the function being annotated.
- The annotation returned by the NoopProvider does not indicate whether the function has any dependencies or not.
- The annotation returned by the NoopProvider does not provide any useful information for identifying functions with no dependencies.
- The annotation returned by the NoopProvider is inconsistent with the actual annotation of the function being annotated.
- The annotation returned by the NoopProvider does not match the expected annotation format.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 NoopProvider returns a TestCaseResult object with the correct attributes.

Why Needed: This test prevents regression where the annotation process fails to return expected results due to missing or incorrect annotations.

Key Assertions:

- The result has an attribute 'scenario'
- The result has an attribute 'why_needed'
- The result has an attribute 'key_assertions'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 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: The test verifies that the ProviderContract handles an empty code by returning a valid result.

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

Key Assertions:

- The provider should return a non-null TestCaseResult object when given an empty code.
- The TestCaseResult object should contain the expected outcome ('passed').
- The annotate method should be able to handle an empty string as input without raising an error.
- The result should not be None, indicating that the contract handled the empty code successfully.
- The provider's configuration and test case should still work correctly even with empty code.
- The TestCaseResult object should contain the correct nodeid and outcome values.

COVERAGE

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

Scenario: The test verifies that the `provider` handles a `None` context in the `test_provider_handles_none_context` method.

Why Needed: This test prevents potential bugs or regressions where the provider might raise an exception when handling a `None` context, potentially causing unexpected behavior or errors.

Key Assertions:

- The `provider.annotate()` method should return `None` if the input `test` is `None`.
- The `provider.annotate()` method should not raise an exception if the input `test` is `None`.
- The `provider.annotate()` method should set the `outcome` attribute to `'passed'` when the input `test` is `None`.
- The `provider.annotate()` method should not throw any exceptions when the input `test` is `None`.
- The `provider.annotate()` method should update the `result` object with a `code` value of `'passed'` when the input `test` is `None`.
- The `provider.annotate()` method should set the `outcome` attribute to `'passed'` in the `result` object even if the input `test` is `None`, indicating successful annotation.
- The `provider.annotate()` method should not update the `result` object with a different value (e.g., `'failed'`) when the input `test` is `None`.
- The `provider.annotate()` method should maintain its original behavior and return the expected result (`None`) for all inputs, including `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	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: All providers should have an 'annotate' method.

Why Needed: This test prevents a potential bug where the LLM contract does not provide an annotate method for some providers.

Key Assertions:

- The provider has an attribute named 'annotate'.
- The provider has a callable attribute named 'annotate'.
- The provider is of type Config with a model parameter set to 'test'.
- The provider is of type get_provider and has a config object with the provider name as its value.
- The provider's annotate method is callable.

COVERAGE

src/pytest_llm_report/collector.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 `annotate` method of the `GeminiProvider` class is called with a context that exceeds its capacity.

Why Needed: This test prevents a potential issue where the `annotate` method fails due to an insufficient context size.

Key Assertions:

- The `context_size` attribute of the `GeminiProvider` instance should be less than or equal to 1000.
- The `annotate` method should not raise an exception when called with a context that exceeds its capacity.
- The `context` argument passed to the `annotate` method should have a size of 1000 or less.

COVERAGE

src/pytest_llm_report/collector.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: Test that the LiteLLMProvider annotates a missing dependency correctly.

Why Needed: This test prevents a potential bug where the provider does not report an error for missing dependencies.

Key Assertions:

- The annotation should contain an error message indicating the missing dependency (litellm).
- The error message should include the correct installation instructions.
- The annotation should indicate that the test case failed due to 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/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

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/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 annotating records tokens with Gemini provider.

Why Needed: Prevents regressions by verifying token usage and rate limits.

Key Assertions:

- Verify that the annotation of test_login() is successful.
- Check if the limiter has recorded at least one token.
- Verify that the limiter's token count matches the expected value.
- Assert that the limiter does not raise an exception during execution.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Verifies that the LLM provider annotates retries on rate limits correctly.

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

Key Assertions:

- The provider should retry after rate limiting occurs.
- The retry attempts should be limited to a reasonable number of times (e.g. 3-5)
- The provider should retry immediately if it is unable to reach the API due to rate limiting
- The provider should not retry indefinitely in case of temporary network issues
- The retry delay should be sufficient to allow for rate limiting to reset
- The provider should retry after a short delay (e.g. 1-2 seconds) between retries
- The provider should retry with the same client ID and request metadata as before

COVERAGE

src/pytest_llm_report/collector.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 `annotate` method rotates models on the daily limit when used with a Gemini provider.

Why Needed: This test prevents a potential bug where the model rotation is not applied correctly to models that are rotated on the daily limit.

Key Assertions:

- Verify that the `rotate_models_on_daily_limit` method of the Gemini provider rotates models correctly.
- Check if the number of models rotated is equal to the expected value for models rotated on the daily limit.
- Verify that the rotation is applied only once per day, even if multiple providers are used concurrently.
- Test if the `rotate_models_on_daily_limit` method throws an exception when called with a provider that does not support daily limits.
- Check if the model rotation is correctly applied to models that have been rotated on previous days.
- Verify that the rotation is applied only once per day, even if multiple providers are used concurrently and the same model is rotated on different days.
- Test if the `rotate_models_on_daily_limit` method throws an exception when called with a provider that does not support daily limits or has exceeded its limit.
- Check if the number of models rotated exceeds the expected value for models rotated on the daily limit.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	177 lines (ranges: 39-42, 45-46, 48-50, 52-54, 73, 76-78, 81-82, 84, 87-88, 92-93, 95-96, 100-102, 134, 136-139, 141-142, 160-161, 167-168, 171-172, 174, 176-184, 186-187, 200-208, 210, 213-214, 217-222, 225-226, 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 the annotate method skips daily limits on LLM providers.

Why Needed: This test prevents a regression where annotating models with daily limits would cause errors or warnings.

Key Assertions:

- Annotates models without violating daily limit.
- Does not raise an error when annotating within daily limit.
- Skips the annotate process for models that are already within daily limit.
- Updates LLM provider to skip daily limit for annotated models.
- Verifies that the LLM provider does not raise warnings or errors for annotated models.
- Ensures that the annotate method is properly configured to handle 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

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/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 LLM provider's exhausted model recovers after 24 hours.

Why Needed: This test prevents regression where the model does not recover from exhaustion within 24 hours.

Key Assertions:

- The recovered model should have a similar accuracy to its original state before exhaustion.
- The recovered model should have the same number of parameters as the original model.
- The recovered model's loss function should be close to zero after 24 hours.
- The recovered model's top 10 predictions should still match the top 10 predictions of the original model.
- The recovered model's top 5 predictions should still match the top 5 predictions of the original model.
- The recovered model's top 3 predictions should still match the top 3 predictions of the original model.
- The recovered model's top 2 predictions should still match the top 2 predictions of the original model.
- The recovered model's top 1 prediction should still match the top 1 prediction of the original 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	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

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	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 is refreshed after an interval of time.

Why Needed: This test prevents a potential regression where the model list does not refresh after an interval of time.

Key Assertions:

- The `refresh_interval` attribute of each provider is set to the expected value.
- Each provider's `refreshed_at` timestamp is updated to the current time.
- All providers' lists are refreshed and their timestamps are up-to-date.
- No provider's list remains stale for more than the expected interval.
- The `refresh_interval` attribute of each provider is correctly set to a positive value.
- Each provider's `refreshed_at` timestamp is correctly updated to the current time.
- All providers' lists are refreshed and their timestamps are up-to-date after an interval of time.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 that LiteLLM provider retries on 401 after refreshing token.

Why Needed: Reason: The current implementation does not retry on 401 errors when the token is refreshed.

Key Assertions:

- Verify that the test case is executed twice, once with a failed token and once with a successful refresh of the token.
- Verify that the first captured API key is 'token-1' and the second one is 'token-2'.
- Confirm that the provider retries on 401 errors when the token is refreshed.

COVERAGE

src/pytest_llm_report/collector.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 LiteLLMProvider annotates completion errors correctly.

Why Needed: This test prevents a regression where the LiteLLMProvider does not surface completion errors when they occur.

Key Assertions:

- The annotation should contain an error message indicating the cause of the completion error.
- The error message should be 'boom' as expected.
- The annotation should also include the line number and function name where the error occurred.
- The annotation should not ignore completion errors without providing any meaningful information.
- The test case should pass even if a completion error occurs in a different part of the code.
- The LiteLLMProvider should be able to handle completion errors in annotations correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 the LiteLLMProvider from accepting and processing invalid key_assertions payloads.

Key Assertions:

- response_data must be a dictionary
- response_data must contain a 'key_assertions' key
- response_data must contain a list of 'key_assertions' values

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 test verifies that the LiteLLMProvider annotates missing dependencies correctly.

Why Needed: This test prevents a bug where the provider fails to report missing dependencies when they are required by other nodes in the pipeline.

Key Assertions:

- The annotation returned by the annotate method contains an error message indicating what dependency is missing.
- The annotation returns 'litellm not installed. Install with: pip install litellm' as a string value for the error property.
- The annotation includes the correct provider name ('litellm') in its error message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: Prevents regressions caused by missing or incorrect annotation logic, ensuring consistent behavior across different responses.

Key Assertions:

- The `scenario` field is set to 'Checks login'.
- The `why_needed` field is set to 'Stops regressions'.
- The `key_assertions` list contains the expected assertions ['status ok', 'redirect'].
- The `confidence` field is set to 0.8, indicating a high level of certainty in the annotation result.
- The `model` field is set to 'gpt-4o'.
- The `messages` dictionary contains the expected messages ['system', 'def test_login()'].
- The `content` key within the first message matches the expected 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	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 API keys are not properly passed through to the completion function.

Key Assertions:

- The API key is set to `static-key-placeholder` before passing it to the completion function.
- The API key is retrieved from the environment variable `TEST_KEY` if it exists, otherwise it defaults to `static-key-placeholder`.
- The API key is not modified or overridden by any other factors during the 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/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: This test prevents a potential issue where the provider does not detect installed modules, potentially leading to unexpected behavior or errors.

Key Assertions:

- The `is_available()` method returns True when the module is detected.
- The `litellm` module is set as the default module in the system.
- The `sys.modules` dictionary contains the `litellm` module.
- The `LiteLLMProvider` instance has an `is_available()` attribute that returns True.
- The test passes when the `litellm` module is installed and available in the system.
- The test fails when the `litellm` module is not installed or not available in the system.

COVERAGE

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

Why Needed: The test prevents a bug where the LLM provider does not refresh tokens properly, leading to unexpected behavior or errors.

Key Assertions:

- Verify that the `litellm_token_refresh_command` is set correctly to 'get-token'.
- Check if the `litellm_token_refresh_interval` is set correctly to 3600 seconds (1 hour).
- Assert that the API key captured from the subprocess output matches the expected value.
- Verify that the `FakeLiteLLMResponse` object created by `fake_completion` has the correct JSON data.
- Check if the `subprocess.CompletedProcess` object returned by `fake_run` has a return code of 0 and no error message.
- Assert that the `litellm_token_refresh_command` attribute of the `SimpleNamespace` instance is set correctly to 'get-token'.
- Verify that the `litellm_token_refresh_interval` attribute of the `Config` object is set correctly to 3600 seconds (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: Testing the `annotate_fallbacks_on_context_length_error` method of the `OllamaProvider` class.

Why Needed: This test prevents a potential regression where the `annotate_fallbacks_on_context_length_error` method fails when the context length is too long.

Key Assertions:

- The `annotate_fallbacks_on_context_length_error` method should not raise an error when the context length is too long.
- The `annotate_fallbacks_on_context_length_error` method should return a meaningful error message when the context length is too long.
- The `annotate_fallbacks_on_context_length_error` method should handle the case where the context length is too long without raising an exception or 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/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: The test verifies that the Ollama provider correctly annotates a call with an error.

Why Needed: This test prevents regression in handling call errors, ensuring that the annotation is accurate and informative.

Key Assertions:

- The annotation should include an error message indicating that the call failed after retries.
- The error message should be 'Failed after 2 retries. Last error: boom'.
- The annotation should not include any additional information about the system prompt.
- The annotation should only report the last error encountered during the call.
- The annotation should use the correct exception type ('boom') to represent the error.
- The annotation should be able to handle cases where the provider is unable to make a successful call.

COVERAGE

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

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/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 the full annotation flow of Ollama provider with mocked HTTP.

Why Needed: This test prevents authentication bugs that may occur when annotating user data.

Key Assertions:

- Check if the status of the response is OK (200).
- Validate the token in the response.
- Assert True as the expected result of the annotation.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 regression in the Ollama provider's _call_ollama method when used with a system prompt.

Key Assertions:

- The captured URL matches the expected URL for the Ollama provider.
- The captured JSON contains the correct model and prompt information.
- The captured JSON does not contain the 'stream' key, as requested by the test.
- The timeout is set to 60 seconds, as specified in the configuration.
- The response from the API call is 'test response', which matches 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/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: The test verifies that the Ollama provider uses the default model when not specified.

Why Needed: This test prevents a bug where the model is not set to 'llama3.2' even if it's not provided in the config.

Key Assertions:

- the captured JSON response contains a 'model' key with value 'llama3.2'
- the provider._call_ollama method does not raise an exception when the model is not specified
- the provider._call_ollama method sets the captured JSON response to include the default model
- the captured JSON response includes the correct format for the default model
- the provider._call_ollama method does not update the config with the provided model
- the provider._call_ollama method does not raise an exception when the model is not specified

COVERAGE

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

Why Needed: This test prevents a bug where the provider incorrectly returns True for a server that is not running.

Key Assertions:

- The method _check_availability() of the provider class should return False when the server is unavailable.
- The provider should raise a ConnectionError when trying to get data from the server.
- The provider should set its internal state to False indicating the server is not available.
- The provider's internal state should be updated correctly after calling _check_availability() and before returning False.
- The provider's return value should match the expected False value for a server that is unavailable.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	6 lines (ranges: 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: Test that the Ollama provider returns False for non-200 status codes.

Why Needed: This test prevents a potential bug where the provider incorrectly assumes all requests are successful (status code 200) when they may not be.

Key Assertions:

- The provider's _check_availability method should return False for a status code other than 200.
- The provider's _check_availability method should raise an exception or return None if the status code is not 200.
- The provider's _check_availability method should check the response object's status_code attribute to determine its status code.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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 potential bug where the provider does not check for availability before returning a response.

Key Assertions:

- The '/api/tags' URL is present in the provided request.
- A successful HTTP status code (200) is returned by the /api/tags endpoint.
- The provider's _check_availability method returns True.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 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

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	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: Test that an `OllamaProvider` instance throws an error when parsing a non-JSON response.

Why Needed: To prevent the test from passing and potentially allowing invalid responses to pass through, which could lead to incorrect LLM provider behavior or errors.

Key Assertions:

- The `_parse_response()` method of `OllamaProvider` raises an `AssertionError` with the message 'Failed to parse LLM response as JSON'.
- The error message indicates that the provided string is not a valid JSON.
- The test checks if the provider throws an error when parsing a non-JSON response, which ensures the correct behavior of the OllamaProvider instance.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: Ollama provider rejects invalid key_assertions payloads when parsing responses.

Why Needed: The Ollama provider should raise an error when receiving an invalid 'key_assertions' payload in the response data.

Key Assertions:

- response_data must be a dictionary with a 'key_assertions' key
- response_data must contain a list of strings for 'key_assertions'
- response_data must not contain any other keys besides 'scenario', 'why_needed', and 'error'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	16 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207-209)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 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 function, `test_parse_response_json_in_code_fence`, tests the functionality of extracting JSON from markdown code fences using an Ollama provider.

Why Needed: This test prevents a potential bug where the Ollama provider fails to extract JSON from markdown code fences due to incorrect formatting or missing quotes.

Key Assertions:

- The extracted JSON should be valid and contain only string values.
- The extracted JSON should not contain any null or undefined values.
- The extracted JSON should have a single top-level object with the correct structure.
- The extracted JSON should include a 'text' key with the expected value.
- The extracted JSON should include a 'code' key with the expected value (a string).
- The extracted JSON should not contain any nested objects or arrays that are not properly formatted as strings.
- The extracted JSON should have a single top-level object with the correct structure and no missing quotes or commas.

COVERAGE

src/pytest_llm_report/collector.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 can extract JSON from a plain markdown fence without any language specification.

Why Needed: This test prevents a potential bug where the provider fails to parse JSON responses when there is no specified language.

Key Assertions:

- The response should contain a JSON object with a single key-value pair.
- The JSON object should have a 'data' key with a string value.
- The JSON object should not have any other keys or values.
- The provider should raise an exception if the response is not valid JSON.
- The provider should return None if there is no JSON in the response.
- The provider should ignore non-JSON data in the response.
- The provider should handle cases where the response contains multiple JSON objects.
- The provider should correctly parse JSON from a single JSON object.

COVERAGE

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

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	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: Test CoverageEntry to_dict serialization correctness.

Why Needed: CoverageEntry's to_dict method should be able to serialize the object correctly without any errors.

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 'file_path' value is correct and matches the expected file path.
- The 'line_ranges' value is correct and matches the expected line ranges.
- The 'line_count' value is correct and matches the expected number of lines.
- No other assertions are needed as coverage entry's to_dict method should work correctly 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/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: Test 'test_to_dict' verifies that the `CoverageEntry` class correctly serializes a coverage entry to a dictionary.

Why Needed: This test prevents a potential bug where the `CoverageEntry` class does not serialize all attributes correctly, potentially leading to incorrect data being stored in the serialized dictionary.

Key Assertions:

- The 'file_path' attribute is set to the correct value.
- The 'line_ranges' attribute is set to the correct format (e.g. '1-3, 5, 10-15').
- The 'line_count' attribute is set to the correct value (10 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/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 coverage entry serialization.

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 should be equal to 'src/foo.py'.
- The 'line_ranges' key in the dictionary should be equal to '1-3, 5, 10-15'.
- The 'line_count' key in the dictionary should be equal to 10.
- The 'file_path' value is not a string.
- The 'line_ranges' value is not a string or a list of strings.
- The 'line_ranges' value contains non-digit characters (e.g. ',', '.', '?').
- The 'line_count' value is not an integer.
- The 'line_count' value is less than 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	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 regression where an empty annotation does not have any key assertions.

Key Assertions:

- annotation.scenario == "" (empty string)
- annotation.why_needed == "" (empty string) for regression purposes
- annotation.key_assertions == [] (no key assertions)
- assert annotation.confidence is None (default confidence value)
- assert annotation.error is None (default error message)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 `LlmAnnotation` object can be serialized with all required fields.

Why Needed: This test prevents a potential bug where an annotation is missing some of its necessary fields during serialization.

Key Assertions:

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

COVERAGE

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

Scenario: Test to dictionary with all fields

Why Needed: Prevents loss of annotation details in serialization

Key Assertions:

- Asserts the presence of 'scenario' key in the output dictionary
- Asserts the value of 'confidence' key matches the expected confidence level
- Asserts the presence and correct format of 'context_summary' key
- Asserts that all required fields are present and have valid values

COVERAGE

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

Why Needed: Prevents regression of default report schema version and empty lists.

Key Assertions:

- The 'schema_version' key should be present in the report dictionary with a value equal to SCHEMA_VERSION.
- The 'tests' key should be an empty list.
- The 'warnings' key should not be present in the report dictionary.
- The 'collection_errors' key should not be present in the report dictionary.
- Both 'schema_version' and 'tests' keys should exist in the report dictionary with correct values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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 regression where the report might not include all collection errors due to an incorrect assumption about the number of collection errors.

Key Assertions:

- The length of `collection_errors` in the report should be 1.
- The nodeid of the first error in `collection_errors` should match 'test_bad.py'.
- All errors in `collection_errors` should have a non-empty 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	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 that the ReportRoot class correctly handles warnings in a report.

Why Needed: This test prevents a regression where reports with warnings are not properly identified as such.

Key Assertions:

- The 'warnings' list of the ReportRoot object should contain exactly one warning.
- The code of the first warning in the 'warnings' list should be 'W001'.
- The 'code' attribute of the first warning in the 'warnings' list should match the provided value 'W001'.

COVERAGE

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

Scenario: Tests should be sorted by nodeid in output.

Why Needed: This test prevents a regression where the tests are not sorted correctly by their nodeids, potentially causing issues with report generation and analysis.

Key Assertions:

- The list of nodeids is sorted correctly based on the 'nodeid' attribute of each TestCaseResult.
- Each nodeid appears only once in the sorted list.
- No duplicate nodeids appear in the sorted list.
- The order of nodeids matches the expected order (a, m, z).
- The nodeids are not empty.
- All nodeids have a corresponding test result.
- No nodeid is missing from the sorted list.
- The sorted list contains only unique nodeids.

COVERAGE

src/pytest_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: Testing the `to_dict()` method of `ReportWarning` class to ensure it returns detailed warnings.

Why Needed: This test prevents a potential issue where the detailed warning is missing, potentially leading to coverage issues or other problems.

Key Assertions:

- The `detail` key in the returned dictionary should contain the path '/path/to/file'.
- The value of the `detail` key should be exactly '/path/to/file'.
- If the detail path is different, an assertion error should be raised.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: The test verifies that a ReportWarning object is created without including its detail information.

Why Needed: This test prevents a warning about missing detailed information from being reported.

Key Assertions:

- A ReportWarning object is created with the given code and message.
- The 'detail' key is not included in the dictionary representation of the ReportWarning object.
- The 'message' key is present but does not include any additional details.

COVERAGE

src/pytest_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: Test that RunMeta has aggregation fields.

Why Needed: Prevents regression in aggregation field functionality.

Key Assertions:

- The 'run_id' key should be present and equal to 'run-123'.
- The 'run_group_id' key should be present and equal to 'group-456'.
- The 'is_aggregated' key should be True.
- The 'aggregation_policy' key should be 'merge'.
- The 'run_count' key should be 3.
- The length of the 'source_reports' list 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 LLM fields are excluded when annotations not enabled.

Why Needed: Prevents regression where LLM fields are included due to annotation settings being disabled.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	29 lines (ranges: 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: Test LLM traceability fields are included when enabled.

Why Needed: Prevents regression in LLM model tracing functionality.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	40 lines (ranges: 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' field is included in aggregated runs, potentially hiding important information.

Key Assertions:

- The 'source_reports' key is present in the dictionary.
- The value of 'is_aggregated' is set to False.
- The 'source_reports' key is not present in the dictionary for non-aggregated runs.
- The 'source_reports' field does not contain any values.
- The 'is_aggregated' field remains True even after removing 'source_reports'.
- Removing 'source_reports' from a non-aggregated run still returns a valid 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	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 when using legacy fields without specifying them.

Key Assertions:

- The 'git_sha' field should be set to 'abc1234'.
- The 'git_dirty' field should be set to True.
- The 'repo_version' field should be set to '1.0.0'.
- The 'repo_git_sha' field should be set to 'abc1234'.
- The 'repo_git_dirty' field should be set to True.
- The 'plugin_git_sha' field should be set to 'def5678'.
- The 'plugin_git_dirty' field should be set to False.
- The 'config_hash' field should be set to 'def5678'.
- The length of the 'source_reports' 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	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 RunMeta includes required run status fields.

Why Needed: This test prevents a potential regression where the RunMeta class does not include all necessary fields for a successful run.

Key Assertions:

- The 'exit_code' field is set to 1 as expected.
- The 'interrupted' field is True as expected.
- The 'collect_only' field is True as expected.
- The 'collected_count' field equals the specified value (10) as expected.
- The 'selected_count' field equals the specified value (8) as expected.
- The 'deselected_count' field equals the specified value (2) as expected.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: Verifies the schema version is in semver format.

Why Needed: Prevents regression where the schema version is not in semver format, potentially causing issues with compatibility or validation.

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 three parts of the schema version should have the same length

COVERAGE

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

PASSED `tests/test_models.py::TestSchemaVersion::test_schema_version_in_report_root` 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	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: Test that `CoverageEntry.to_dict()` correctly serializes a CoverageEntry instance.

Why Needed: This test prevents a potential bug where the serialized representation of a CoverageEntry instance is incorrect, potentially leading to unexpected behavior or errors in downstream applications.

Key Assertions:

- The 'file_path' key in the serialized dictionary matches the expected value.
- The 'line_ranges' key in the serialized dictionary matches the expected value.
- The 'line_count' key in the serialized dictionary matches the expected value.
- The 'to_dict()' method of a CoverageEntry instance returns a dictionary with the correct keys and values.
- The 'to_dict()' method of a CoverageEntry instance correctly serializes the file path, line ranges, and line count properties.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: The test verifies that the `to_dict` method of LlmAnnotation returns a dictionary with all required fields.

Why Needed: This test prevents regression by ensuring that the minimal annotation format includes the necessary keys.

Key Assertions:

- assert 'scenario' in d
- assert 'why_needed' in d
- assert 'key_assertions' in d

COVERAGE

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

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	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 regression in coverage entry serialization.

Key Assertions:

- The 'file_path' key is present and has the correct value.
- The 'line_ranges' key is present and has the correct format (e.g., '1-3, 5, 10-15').
- The 'line_count' key is present and has the correct value.
- A dictionary with all required keys is returned by `to_dict()`.
- The values of each key are as expected (i.e., string representations).

COVERAGE

src/pytest_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 required fields.

Why Needed: This test prevents a regression where the minimal result is missing some necessary information.

Key Assertions:

- The 'nodeid' field should match the expected node id.
- The 'outcome' field should be set to 'passed'.
- The 'duration' field should be set to 0.0 (i.e., no execution time).
- The 'phase' field should be set to 'call'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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 `test_result_with_coverage` verifies that the `TestCaseResult` object includes a coverage list.

Why Needed: This test prevents regression in cases where the coverage report is not included in the result.

Key Assertions:

- The length of the 'coverage' key in the `result.to_dict()` dictionary should be 1.
- The value of the 'file_path' key in the first element of the 'coverage' list should be 'src/foo.py'.
- All elements in the 'coverage' list should have a non-empty 'file_path' attribute.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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 case `test_result_with_llm_opt_out` verifies the inclusion of `llm_opt_out` flag in the test result.

Why Needed: The presence of this flag prevents regression where LLM opt-out was not correctly handled by the test.

Key Assertions:

- The `llm_opt_out` field is present and set to True in the test result dictionary.
- The value of `llm_opt_out` is indeed `True` as per the expected behavior.
- The test case ensures that the flag is correctly set even when LLM opt-out is enabled.
- The presence of this flag does not prevent other tests from passing or failing independently.
- The `llm_opt_out` field is included in the JSON representation of the test result.
- The dictionary structure and key-value pairs are consistent with the expected format.
- The assertion checks for the correct value of `llm_opt_out` within 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.py::TestTestCaseResult::test_result_with_rerun 1ms 3

AI ASSESSMENT

Scenario: Test 'Result with reruns' verifies that the 'rerun_count' field is correctly populated in the TestCaseResult object.

Why Needed: This test prevents a regression where the 'rerun_count' field might be missing or incorrect when the result is rerun.

Key Assertions:

- The 'rerun_count' field should contain the expected value of 2.
- The 'final_outcome' field should still be set to 'passed' even after a rerun.
- The 'rerun_count' field should not be changed when the result is rerun.

COVERAGE

src/pytest_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 verifies that the `result` dictionary does not contain 'rerun_count' and 'final_outcome' keys.

Why Needed: This test prevents a regression where the result dictionary contains these fields when rerunning tests without reruns.

Key Assertions:

- The 'rerun_count' key should be absent from the `result` dictionary.
- The 'final_outcome' key should also be absent from the `result` dictionary.
- The expected output of this test should match the actual output when running tests without reruns.
- When a test is run without reruns, the 'rerun_count' and 'final_outcome' fields are not present in the result dictionary.
- This test ensures that the `result` dictionary does not contain these specific keys when rerunning tests without reruns.
- The presence of these fields in the result dictionary indicates a regression or bug in the code.
- When running tests without reruns, the expected output should be different from the actual output.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`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_with_all_optional_fields verifies that the `to_dict` method includes all optional fields when set.

Why Needed: This test prevents regression in coverage reporting, where missing optional fields can lead to incomplete or inaccurate coverage reports.

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`	`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 to_dict includes artifacts when set.

Why Needed: Prevents regression by ensuring that the report includes all required artifacts even when to_dict is called with an empty list of artifacts.

Key Assertions:

- The length of result['artifacts'] should be equal to 2.
- result['artifacts'][0]['path'] should match 'report.html'.
- All artifact entries in the result dictionary should have a non-empty 'path' 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	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 includes collection_errors when set.

Why Needed: The test prevents a potential bug where the to_dict method does not include collection errors in the report.

Key Assertions:

- len(result['collection_errors']) == 1
- result['collection_errors'][0]['nodeid'] == 'broken_test.py'
- assert isinstance(result['collection_errors'][0], dict)
- assert result['collection_errors'][0].get('message') 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/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 regression where custom metadata is not included in the report.

Key Assertions:

- The 'custom_metadata' key exists and has the expected values.
- The value of 'project' is correct.
- The value of 'environment' is correct.
- The value of 'build_number' is correct.
- Custom metadata is included in the report even when not set.
- Custom metadata is excluded from the report when 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/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

Scenario: The test verifies that the `to_dict` method of a `ReportRoot` object includes an HMAC signature when set.

Why Needed: This test prevents a potential security vulnerability where an attacker could manipulate the report's metadata to evade signature verification.

Key Assertions:

- The `hmac_signature` attribute is present in the resulting dictionary.
- The value of the `hmac_signature` attribute matches the expected value 'signature123'.
- The HMAC signature is included in the dictionary even if it was not set when creating the report.
- The presence of the HMAC signature ensures that signature verification can still be performed correctly even with malicious metadata.
- The test covers all possible scenarios where an HMAC signature might be present or absent.
- The `to_dict` method does not modify the original report object, ensuring that the integrity of the data is preserved.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: Verifies that the `to_dict` method of `ReportRoot` includes a SHA-256 hash when set.

Why Needed: Prevents a regression where a report without a SHA-256 hash is incorrectly reported as having one.

Key Assertions:

- The `sha256` attribute of the `ReportRoot` instance is accessed and compared to its value in the resulting dictionary.
- A TypeError is raised if the `to_dict` method is called on an empty report.
- The SHA-256 hash is included in the resulting dictionary with the correct format (e.g., 'abcdef1234567890')
- The SHA-256 hash is correctly calculated and stored in the report instance.
- A ValueError is raised if the `sha256` attribute is not a string or None.
- The `to_dict` method returns a dictionary with the expected structure for reports without a SHA-256 hash.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/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: This test prevents a regression where the source coverage is not included in the report.

Key Assertions:

- The length of `source_coverage` is 1.
- The value of `file_path` in the first element of `source_coverage` is 'src/mod.py'.
- The file path matches the expected value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: The test verifies that the `to_dict` method of `ReportRoot` includes warnings when set.

Why Needed: This test prevents a regression where warnings are not included in the report even though there is coverage data.

Key Assertions:

- The length of `result['warnings']` should be 1.
- The value of `result['warnings'][0]['code']` should be 'W001'.
- The code 'W001' should be present in the warnings list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 regression in test output by including coverage total percent.

Key Assertions:

- The `coverage_total_percent` key is present and has the correct value (85.5).
- The `passed`, `failed`, and `skipped` values are correctly calculated based on the provided counts.
- The `total` count matches the expected total of 10.
- The `Summary` object's attributes (`passed`, `failed`, `skipped`) match the corresponding values in the test 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	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: The test verifies that the `to_dict` method of a `Summary` object excludes the `coverage_total_percent` field when it is `None`.

Why Needed: This test prevents a regression where the total coverage percentage is included in the summary even though there are no tests.

Key Assertions:

- The `summary.to_dict()` method does not include the `coverage_total_percent` key if it is `None`.
- The `to_dict` method excludes the `coverage_total_percent` field when it is `None`.
- When `summary.total` and `summary.passed` are both `None`, the `to_dict` method should return 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/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 reporting 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.

Key Assertions:

- The 'captured_stderr' key should be present in the test result dictionary.
- The value of 'captured_stderr' should match the expected captured stderr string.
- If no captured stderr is provided, the 'captured_stderr' key should not be present in the test result dictionary.
- If an empty string is passed as captured stderr, it should be considered a failure.
- If an exception occurs during the test, it should be included in the captured stderr message.
- The captured stderr message should contain the expected error output.
- The captured stderr message should not be empty or null.
- The captured stderr message should not exceed 1024 characters.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: Prevents a potential bug where the captured stdout is not included in the result.

Key Assertions:

- The 'captured_stdout' key should be present in the test_result dictionary.
- The value of 'captured_stdout' should match the expected captured stdout.
- If no captured stdout is provided, the 'captured_stdout' key should not be present in the result.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: This test prevents a bug where the 'to_dict' method does not include the parameter summary in the output.

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 'to_dict' method includes all required keys in the output.
- Without parameter summary, the test would fail due to missing 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	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 potential bug where the 'requirements' key is missing from the test result dictionary, causing it to fail or produce incorrect results.

Key Assertions:

- The 'requirements' key should be present in the test result dictionary.
- The 'requirements' key should contain the specified 'REQ-001' and 'REQ-002' values.
- If no 'requirements' are provided, the test result dictionary should still contain the required keys ('nodeid', '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/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.

Why Needed: This test prevents a potential regression where the default configuration might be incorrectly set, potentially affecting the behavior of the LLM.

Key Assertions:

- cfg.provider == 'none'
- cfg.llm_context_mode == 'minimal'
- cfg.llm_max_tests == 0
- cfg.llm_max_retries == 10
- cfg.llm_context_bytes == 32000
- cfg.llm_context_file_limit == 10
- cfg.llm_requests_per_minute == 5
- cfg.llm_timeout_seconds == 30
- cfg.llm_cache_ttl_seconds == 86400
- cfg.include_phase == 'run'
- cfg.aggregate_policy == 'latest'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 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: Verifies whether the LLM is enabled based on the provider.

Why Needed: Prevents a potential bug where the LLM is incorrectly enabled for some providers.

Key Assertions:

- The `is_llm_enabled` method returns False when the provider is 'none'.
- The `is_llm_enabled` method returns True when the provider is 'ollama'.
- A new configuration object without a provider is considered disabled and should return False.
- Setting the provider to 'ollama' after creating an empty config object should enable 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`	`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

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

Why Needed: This test prevents a potential bug where the validation fails with incorrect error messages for an invalid include phase.

Key Assertions:

- The function `cfg.validate()` should return a list containing exactly one error message.
- The error message should contain 'Invalid include_phase 'lunch_break'.
- The error message should be present in the first element of the returned 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	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

Scenario: Testing the `validate()` method for an invalid provider.

Why Needed: This test prevents a potential bug where an invalid provider is passed to the configuration, causing unexpected behavior or errors.

Key Assertions:

- The `validate()` method returns a list of error messages.
- At least one error message is present in the list.
- The first error message contains the string 'Invalid provider'.
- The error message mentions the invalid provider name.

COVERAGE

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

Why Needed: This test prevents a potential bug where the configuration is not properly validated against numeric ranges, potentially leading to unexpected behavior or errors.

Key Assertions:

- cfg.validate() returns an error message indicating that llm_context_bytes must be at least 1000
- llm_context_bytes should be greater than or equal to 1000
- cfg.validate() returns an error message indicating that llm_max_tests must be positive
- llm_max_tests should be a non-negative integer
- cfg.validate() returns an error message indicating that llm_requests_per_minute must be at least 1
- llm_requests_per_minute should be greater than or equal to 1
- cfg.validate() returns an error message indicating that llm_timeout_seconds must be at least 1
- llm_timeout_seconds should be greater than or equal to 1

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: The test prevents potential issues that could arise from invalid or malformed configurations.

Key Assertions:

- A `Config` object is created and initialized without any errors.
- The `validate` method is called on the `Config` object, and no errors are reported.
- Any configuration options provided to the `Config` constructor do not cause any validation errors.

COVERAGE

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

Why Needed: This test prevents a regression where the aggregate directory, policy, run ID, and group ID are not set to their default values when loading configuration.

Key Assertions:

- The aggregate directory is set to 'aggr_dir'.
- The aggregate policy is set to 'merge'.
- The aggregate run ID is set to 'run-123'.
- The aggregate group ID is set to 'group-abc'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 regression where the LLM report generation fails due to missing pyproject.toml file.

Key Assertions:

- The 'llm_max_retries' option is set to 10 by default.
- The 'llm_report_html', 'llm_report_json', and 'llm_report_pdf' options are not set.
- The 'llm_evidence_bundle', 'llm_dependency_snapshot', 'llm_requests_per_minute', and 'llm_aggregate_dir' options are set to their default values.
- The 'llm_coverage_source' option is set to its default value.
- The 'llm_max_retries' option is not affected by the presence of pyproject.toml.
- The 'llm_provider', 'llm_model', and 'llm_context_mode' options are not affected by the presence of pyproject.toml.

COVERAGE

src/pytest_llm_report/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 bug where the coverage source is not correctly set to 'cov_dir'.

Key Assertions:

- The value of `cfg.llm_coverage_source` should be equal to 'cov_dir'.
- The value of `cfg.llm_coverage_source` should match the expected value.
- The test checks if the coverage source is correctly set to 'cov_dir' before asserting it.
- The test verifies that the coverage source is not set to an invalid value.
- The test ensures that the coverage source is updated correctly after setting it.
- The test validates that the coverage source is correctly stored in the configuration.
- The test checks if the coverage source is correctly retrieved from the configuration.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 `load_defaults` test loads a default configuration without any options.

Why Needed: Prevents regression when no configuration options are provided.

Key Assertions:

- The `cfg.provider` attribute should be set to 'none'.
- The `cfg.report_html` attribute should be set to None.
- The test loads a default configuration without any provider or report settings.

COVERAGE

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

Why Needed: This test prevents a bug where pyproject.toml overrides CLI options, potentially causing unexpected behavior or errors in the application.

Key Assertions:

- The `--pyproject` option is overridden by the `--pyproject.toml` option.
- The `--load-configuration` option is not overridden by any other option.
- The `--python-version` option is preserved and not affected by CLI options.
- The `--no-deps` option is preserved and not affected by CLI options.
- The `--color` option is preserved and not affected by CLI options.
- The `--format` option is preserved and not affected by CLI options.
- The `--python-version` option has the same value as before (e.g., 3.8).
- The `--no-deps` option has the same value as before (e.g., 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	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` 2ms 3

AI ASSESSMENT

Scenario: Verify that the test_load_from_cli_provider_override function in tests/test_options.py correctly overrides configuration settings provided by a CLI provider.

Why Needed: This test prevents bugs where the default configuration of the application is overridden by a custom configuration provided through the CLI provider.

Key Assertions:

- The pyproject.toml file is created with the correct content.
- The CLI provider option overrides the default configuration settings.
- The test_load_from_cli_provider_override function correctly loads the customized configuration.
- The application's configuration is not affected by the overridden configuration provided through the CLI provider.
- The custom configuration override is saved to the pyproject.toml file.
- The test_load_from_cli_provider_override function checks for and updates the configuration in the pyproject.toml file.
- The test_load_from_cli_provider_override function correctly loads the customized configuration from 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, 414-415, 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 in the CLI is set to 2.

Why Needed: This test prevents a potential bug where the `llm_max_retries` option is not correctly set when loading configuration from the CLI.

Key Assertions:

- The value of `llm_max_retries` should be equal to 2.
- The `llm_max_retries` option in the CLI should have been successfully updated with a value of 2.
- The test should fail if the `llm_max_retries` option is not set correctly when loading configuration from the CLI.

COVERAGE

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

AI ASSESSMENT

Scenario: Verify that the `test_load_from_pyproject` test verifies the ability to load configuration values from a `pyproject.toml` file.

Why Needed: This test prevents regression in the case where the `load_config` function is called with a `pyproject.toml` file provided as an argument, without any additional arguments.

Key Assertions:

- The `test_load_from_pyproject` test verifies that the `load_config` function correctly loads configuration values from a `pyproject.toml` file.
- The `load_config` function should be able to read and parse the contents of the `pyproject.toml` file without any issues.
- The `load_config` function should throw an exception if the `pyproject.toml` file is missing or corrupted.
- The `test_load_from_pyproject` test verifies that the `load_config` function handles the case where the `pyproject.toml` file is provided as a string argument correctly.
- The `load_config` function should be able to load configuration values from a `pyproject.toml` file with missing or corrupted data without any issues.
- The `test_load_from_pyproject` test verifies that the `load_config` function throws an exception when it encounters invalid or unsupported configuration data in the `pyproject.toml` file.
- The `load_config` function should be able to handle nested configuration structures within the `pyproject.toml` file 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	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 set to 'deps.json' when running CLI with this configuration.

Why Needed: This test prevents a potential regression where the `llm_dependency_snapshot` option is not set correctly, causing the dependency snapshot report to be incorrect.

Key Assertions:

- The value of `llm_dependency_snapshot` in the configuration file should be 'deps.json'.
- The `llm_dependency_snapshot` option should be present in the configuration file.
- The value of `llm_dependency_snapshot` should match the expected value 'deps.json' when loaded from the configuration file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 correct evidence bundle is reported when CLI option is set.

Why Needed: This test prevents a potential regression where the correct evidence bundle might not be reported if the CLI option is not set correctly.

Key Assertions:

- The value of `report_evidence_bundle` in the configuration file is set to `bundle.zip` when the CLI option is set to `llm_evidence_bundle = 'bundle.zip'`.
- The correct evidence bundle should be reported in the configuration file.
- The value of `report_evidence_bundle` in the configuration file does not match the expected value when the CLI option is not set correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: Verify that the test CLI overrides the default report format to 'output.json'.

Why Needed: Prevents a regression where the default report format is not set correctly.

Key Assertions:

- cfg.report_json is set to 'output.json' after creating mock config.
- assert cfg.report_json == 'output.json' after setting llm_report_json option.
- mock._make_mock_config(tmp_path) returns a mock config with the correct report format.
- load_config(mock) sets the default report format correctly based on the mock config.
- cfg.report_json is not set to an empty string or None before the test.
- assert cfg.report_json is not None and not empty after setting llm_report_json option.
- mock.option.llm_report_json is set to 'output.json' in the mock config.
- cfg.report_json is updated correctly when the llm_report_json option 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/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: Verify that the `test_cli_report_pdf` test case sets the expected report PDF path correctly.

Why Needed: This test prevents a potential bug where the reported PDF file is not set to the correct location.

Key Assertions:

- The value of `cfg.report_pdf` is set to `output.pdf` after calling `mock.option.llm_report_pdf = 'output.pdf'`.
- The expected report PDF path is `output.pdf` as specified in the test case.
- The `load_config(mock)` call sets the correct value for `cfg.report_pdf`.
- The `assert cfg.report_pdf == 'output.pdf'` line checks if the actual value matches the expected one.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: Tests the `validate` method of the `Config` class to ensure it correctly identifies and reports invalid token output formats.

Why Needed: Prevents a potential bug where an invalid token output format is silently ignored, potentially leading to coverage issues or incorrect test results.

Key Assertions:

- The `validate()` method returns errors containing the string 'litellm_token_output_format' for any invalid token output formats.
- Any error messages returned by the `validate()` method contain the exact phrase 'litellm_token_output_format'.
- The `validate()` method reports an error for each invalid token output format encountered during validation.
- The test asserts that at least one error message contains the string 'litellm_token_output_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`	`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: This test prevents a potential bug where the token refresh interval is set to a value that does not provide sufficient time for the application to handle token refresh requests.

Key Assertions:

- litellm_token_refresh_interval must be at least 60
- litellm_token_refresh_interval must be an integer
- litellm_token_refresh_interval should be greater than or equal to 1

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 a potential bug where an invalid or missing config is passed to the `validate` method, causing it to fail and return an error.

Key Assertions:

- The `cfg.validate()` method should not return any errors when a valid configuration is provided.
- A valid LiteLLM provider (e.g. 'litellm') should be used without any issues.
- The `litellm_token_output_format` parameter should be set to 'text' as expected.
- The `litellm_token_refresh_interval` parameter should be set to 3600 seconds (1 hour) as expected.
- No errors should be raised when the configuration is valid and can be processed without issues.
- The validation process should not hang or freeze due to an invalid config.
- A valid configuration should be able to be used for further processing without any regressions.

COVERAGE

src/pytest_llm_report/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: Tests the ability to load aggregate_include_history from pyproject.toml.

Why Needed: Prevents a potential bug where the test fails if the 'aggregate_include_history' is not present in the pyproject.toml file.

Key Assertions:

- The 'pyproject.toml' file should contain the 'aggregate_include_history' key with the expected value.
- The 'aggregate_include_history' value should be a string or bytes object.
- The 'aggregate_include_history' value should not be empty.
- The 'aggregate_include_history' value should only contain characters from the set of allowed characters (e.g., letters, digits, underscores).
- The 'aggregate_include_history' value should not contain any whitespace characters.
- The 'pyproject.toml' file should exist and be a valid pyproject.toml file.
- The test should fail if the 'aggregate_include_history' is missing or empty 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, 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 regression in cases where the `load_aggregate_policy` function is unable to find or read the PyProject file.

Key Assertions:

- The `load_aggregate_policy` function should be able to write the loaded aggregate policy to a new file with the same name as the original PyProject file.
- The `load_aggregate_policy` function should not raise any exceptions when loading an empty or invalid PyProject file.
- The `load_aggregate_policy` function should correctly handle cases where the PyProject file is missing or corrupted.
- The `load_aggregate_policy` function should be able to load aggregate policies from multiple PyProject files in a single test run.
- The `load_aggregate_policy` function should preserve the original directory structure and file names of the PyProject file when loading it.
- The `load_aggregate_policy` function should not modify the contents or metadata of the loaded aggregate policy.
- The `load_aggregate_policy` function should return an empty list for a PyProject file that does not contain any aggregate policies.

COVERAGE

src/pytest_llm_report/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 correctly loads all configuration keys from a single pyproject.toml file.

Why Needed: This test prevents regression in case of changes to the pyproject.toml file structure, where the 'load_all_config_keys' function needs to be updated to handle new configuration keys.

Key Assertions:

- The 'load_all_config_keys' function should correctly load all configuration keys from the specified pyproject.toml file.
- The function should not raise any errors when loading a file with missing or invalid configuration keys.
- All configuration keys in the loaded file should be present and accessible through the 'config' dictionary.
- The function should handle nested configurations correctly, i.e., load sub-keys from parent directories as well.
- Any configuration keys that are not explicitly listed in the pyproject.toml file should be ignored or silently skipped by the function.
- The function's output should match the expected configuration keys and values.
- The 'load_all_config_keys' function should be able to handle a large number of configuration keys without performance 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	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 dependencies or other system functions.

Key Assertions:

- The `cache_dir` value in the `pyproject.toml` file matches the expected path.
- The `cache_dir` value is set to the correct directory based on the `python_version` and `platform` values.
- The `cache_dir` value is not empty or None, ensuring it's a valid directory path.
- The `cache_dir` value is relative to the current working directory, rather than an absolute path.
- The `cache_dir` value is set correctly for both Windows and Unix-like systems.
- The `cache_dir` value does not contain any invalid characters or special regex patterns.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 in `pyproject.toml` is loaded correctly.

Why Needed: This test prevents a potential bug where the cache TTL seconds are not loaded from `pyproject.toml`.

Key Assertions:

- The value of `cache_ttl_seconds` in `pyproject.toml` is set to a valid integer.
- The `cache_ttl_seconds` value is greater than or equal to 0.
- The `cache_ttl_seconds` value is less than the maximum allowed value (3600 seconds).
- The file path of the `pyproject.toml` file exists and is not empty.
- The `pyproject.toml` file is written with a valid format.
- The `cache_ttl_seconds` value in `pyproject.toml` is correctly formatted as an integer.
- The `cache_ttl_seconds` value does not exceed the maximum allowed value (3600 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/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: Test that `load_capture` function loads correct output for failed capture.

Why Needed: Prevents regression in case of failed capture during testing.

Key Assertions:

- The `load_capture` function should be able to load the captured output from pyproject.toml correctly.
- The captured output should match what is expected in the test.
- The error message or output should indicate that capture was not successful.
- The `pyproject.toml` file should contain a section named `capture_failed_output` with the correct content.
- The function should raise an exception when the pyproject.toml does not contain the required section.
- The test should fail if the captured output is empty or contains unexpected data.
- The `load_capture` function should be able to handle different types of capture (e.g., console, file).
- The function should provide a meaningful error message when loading failed capture.

COVERAGE

src/pytest_llm_report/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 the `load_capture_output_max_chars` function in `test_options_coverage.py` can load capture output with a maximum character count.

Why Needed: This test prevents regression where the function fails to load capture output with a maximum character count.

Key Assertions:

- The captured output is not longer than 1000 characters.
- The 'max_chars' setting in pyproject.toml is applied correctly.
- The function can handle large input files without errors.

COVERAGE

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

Scenario: Verify that the `load_context_bytes` function correctly loads context bytes from a PyProject file.

Why Needed: This test prevents regression where the function fails to load context bytes in certain cases, potentially causing coverage issues.

Key Assertions:

- The function writes the correct content to the `context_bytes` field in the PyProject.toml file.
- The function correctly handles different types of data (e.g., strings, integers) in the `context_bytes` field.
- The function does not throw any exceptions when encountering invalid or missing data in the PyProject.toml file.
- The `load_context_bytes` function is able to read and write bytes correctly using the `b` type hint.
- The function uses the correct encoding for the context bytes (e.g., UTF-8, binary) as specified in the PyProject.toml file.
- The function does not modify any external state or data structures during execution.

COVERAGE

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

AI ASSESSMENT

Scenario: Verify that the `load_context_exclude_globs` option is correctly loaded from pyproject.toml.

Why Needed: This test prevents a potential bug where the `load_context_exclude_globs` option is not properly loaded, potentially leading to incorrect context exclusion.

Key Assertions:

- The `pyproject.toml` file contains the correct value for the `load_context_exclude_globs` option.
- The `load_context_exclude_globs` option is correctly set to `True` in the pyproject.toml file.
- The test verifies that the `load_context_exclude_globs` option has a valid value.
- The `pyproject.toml` file does not contain any invalid or unexpected values for the `load_context_exclude_globs` option.
- The `load_context_exclude_globs` option is correctly set to `True` in the pyproject.toml file, regardless of the project's configuration.
- The test verifies that the `load_context_exclude_globs` option has a value of `True` or `False` 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	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 loaded correctly from the `pyproject.toml` file.

Why Needed: This test prevents a potential bug where the `context_file_limit` setting is not loaded, potentially leading to incorrect behavior in the Pytest runner or other dependencies.

Key Assertions:

- The `context_file_limit` setting is present and readable from the `pyproject.toml` file.
- The value of `context_file_limit` matches one of the expected values (e.g. '1000000', '2000000', etc.).
- The `context_file_limit` setting is not empty or zero.
- The `context_file_limit` setting is a string that can be parsed as an integer.
- The value of `context_file_limit` does not exceed the maximum allowed value (1000000 for Pytest 6.2.5).
- The value of `context_file_limit` is greater than or equal to the minimum allowed value (2000000 for Pytest 6.1.3).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 2ms 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 when using the `--include-glob` flag.

Key Assertions:

- The contents of the `pyproject.toml` file are correct.
- The `context_include_globs` setting is set to the expected value.
- The test environment includes the specified glob patterns.
- The test environment does not include any excluded glob patterns.
- The `--include-glob` flag is applied correctly and the setting is loaded.
- The `pyproject.toml` file is written with the correct 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/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 `pyproject.toml`.

Why Needed: Prevents a potential bug where the `hmac_key_file` is not loaded due to an incorrect or missing `pyproject.toml` file.

Key Assertions:

- The contents of `pyproject.toml` should contain the expected `hmac_key_file` path.
- The `hmac_key_file` value in `pyproject.toml` should be a string representing the actual file path.
- The `pyproject.toml` file should not be empty or missing any required sections.
- The `hmac_key_file` value should be present and correctly formatted (e.g., `path/to/hmac_key_file`)
- The `pyproject.toml` file should have the correct permissions (e.g., `read-only` for the test user)
- The `hmac_key_file` path should be relative or absolute and match the expected format (e.g., `/path/to/file`)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 the inclusion of `include_param_values` in the Pyproject.toml file.

Why Needed: This test prevents a potential issue where include_param_values is not loaded correctly from pyproject.toml, potentially leading to coverage regression.

Key Assertions:

- The contents of pyproject.toml should contain `include_param_values` as specified in the requirements.
- The file path for pyproject.toml should be correct and accessible.
- The `include_param_values` value should be correctly formatted according to the specifications.
- No additional include directories or parameters should be present without their values.
- The test should fail if `include_param_values` is missing from pyproject.toml.
- The test should pass if `include_param_values` is correctly loaded and formatted in pyproject.toml.

COVERAGE

src/pytest_llm_report/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 issue where the `include_phase` is not properly 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 number of lines.
- The `include_phase` section matches the expected format (e.g., `#include `).
- The `include_phase` section is correctly formatted and does not contain any syntax errors.
- The `include_phase` section contains only valid Python code (e.g., no comments or whitespace).

COVERAGE

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

AI ASSESSMENT

Scenario: Verifies that the `include_pytest_invocation` option is correctly loaded from pyproject.toml.

Why Needed: This test prevents a potential bug where the `include_pytest_invocation` option is not detected or is incorrectly interpreted by Pytest.

Key Assertions:

- The contents of the `pyproject.toml` file are correct and match the expected output.
- The `include_pytest_invocation` option is present in the `pyproject.toml` file.
- The `include_pytest_invocation` option is correctly formatted as a string.
- The `include_pytest_invocation` option does not contain any whitespace characters.
- The `include_pytest_invocation` option is not empty.
- The `include_pytest_invocation` option matches the expected value for Pytest.
- The `pyproject.toml` file is written to a temporary directory and its contents are preserved after the test completes.

COVERAGE

src/pytest_llm_report/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: Testing the ability to load invocation_redact_patterns from pyproject.toml

Why Needed: Prevents a potential bug where invocation_redact_patterns are not loaded correctly due to incorrect or missing configuration.

Key Assertions:

- The `pyproject.toml` file is successfully written with the correct content.
- The `invocation_redact_patterns` key in pyproject.toml exists and has the expected value.
- The `invocation_redact_patterns` key contains only valid patterns (e.g., 'redact', 'replace', etc.)
- The `invocation_redact_patterns` key is not empty or None.
- The `pyproject.toml` file is successfully read by the test environment.
- The `invocation_redact_patterns` value in pyproject.toml matches the expected format (e.g., a list of patterns)
- No exceptions are raised during the 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	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` 2ms 3

AI ASSESSMENT

Scenario: Testing the ability to load litellm_api_base from pyproject.toml.

Why Needed: This test prevents regressions where loading litellm_api_base fails due to missing or corrupted configuration files.

Key Assertions:

- The contents of pyproject.toml should be able to be written successfully.
- The file pyproject.toml exists and is not empty.
- The file pyproject.toml can be read successfully using the write_text method.
- The contents of pyproject.toml are correctly formatted according to the PyProject specification.
- No errors or warnings are raised when writing to pyproject.toml.
- The pyproject.toml file is not corrupted and has a valid format.
- The test can be run multiple times without encountering issues due to changes in pyproject.toml.

COVERAGE

src/pytest_llm_report/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` 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	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 bug where the `litellm_token_json_key` value is not properly loaded or is empty when trying to access it.

Key Assertions:

- The `litellm_token_json_key` value should be present in the `pyproject.toml` file and have the correct format.
- The `litellm_token_json_key` value should match the expected string (e.g., 'litellm_token_json_key_value').
- The `litellm_token_json_key` value should not be empty or null.
- The `pyproject.toml` file should contain a `tool litellm` section with the correct key and value.
- The `pyproject.toml` file should have a `tool litellm.litellm_token_json_key` attribute with the correct value.
- The `litellm_token_json_key_value` string should be properly formatted (e.g., enclosed in quotes).

COVERAGE

src/pytest_llm_report/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: Testing the `load_litellm_token_output_format` function with a specific input.

Why Needed: This test prevents regression in case the `litellm_token_output_format` option is not properly configured in the `pyproject.toml` file.

Key Assertions:

- The output format of `litellm_token_output_format` should be 'json'.
- The output format of `litellm_token_output_format` should be 'json' when using the `--format json` flag.
- The output format of `litellm_token_output_format` should not be changed by other options (e.g., `--format xml`).

COVERAGE

src/pytest_llm_report/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` 2ms 3

AI ASSESSMENT

Scenario: Testing the `load_litellm_token_refresh_command` function to ensure it correctly loads a command from the PyProject.toml file.

Why Needed: This test prevents regression where the `load_litellm_token_refresh_command` function fails to load commands from the PyProject.toml file, potentially causing issues with token refresh.

Key Assertions:

- The `load_litellm_token_refresh_command` function should be able to successfully load a command from the PyProject.toml file without raising an exception.
- The loaded command should be a valid Python module (e.g., `litellm.token_refresh` or `litellm.token_refresh.command`)
- The command's name and docstring should match the expected values (if provided in the PyProject.toml file)
- The function should return an empty list if no commands are found in the PyProject.toml file
- The function should raise a `FileNotFoundError` if the specified command 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	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` 2ms 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 causing 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 in the `pyproject.toml` file.
- The value of the `litellm_token_refresh_interval` option matches the expected value (e.g., '1h', '1d', etc.).
- The `litellm_token_refresh_interval` option is not set to an invalid or empty string.
- The `pyproject.toml` file does not contain any other options that might affect the loading of the `litellm_token_refresh_interval` option.
- The `pyproject.toml` file has a valid and consistent structure.
- The test can be run multiple times without encountering issues due to changes 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, 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 that the test_load_malformed_pyproject function handles malformed pyproject.toml correctly.

Why Needed: This test prevents a potential regression where the function would raise an error when encountering invalid pyproject.toml files.

Key Assertions:

- The function should not raise an exception when given a malformed pyproject.toml file.
- The function should fallback to its default configuration instead of raising an error.
- The 'provider' key in the config dictionary should be set to 'none'.
- The test should pass even if the input pyproject.toml is syntactically incorrect (e.g. missing or extra brackets).

COVERAGE

src/pytest_llm_report/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 the `pyproject.toml` file is correctly loaded and used to configure the concurrency limit.

Why Needed: This test prevents a potential bug where the `max_concurrency` setting is not properly loaded or configured, potentially leading to unexpected behavior or errors when using the `concurrent.futures.ThreadPoolExecutor` class.

Key Assertions:

- The content of the `pyproject.toml` file is correct and matches the expected format.
- The `max_concurrency` setting in the `pyproject.toml` file is set to a valid value (e.g., `10` or `100`).
- The `concurrent.futures.ThreadPoolExecutor` class can be used without any issues when the `max_concurrency` setting is properly loaded and configured.
- The test does not encounter any exceptions or errors when using the `concurrent.futures.ThreadPoolExecutor` class with a concurrency limit set to `max_concurrency`.
- The `concurrent.futures` module can be imported without any issues, even if the `max_concurrency` setting is not properly loaded.
- The test passes without any warnings or errors when running on a system with a valid `pyproject.toml` file and a configured concurrency limit.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 `max_tests` setting is not properly loaded, potentially leading to incorrect coverage analysis.

Key Assertions:

- The contents of the `pyproject.toml` file are correct.
- The `max_tests` setting in the `pyproject.toml` file is set correctly.
- The test loads the maximum number of tests specified in the `pyproject.toml` file.
- The coverage analysis includes all tests specified in the `pyproject.toml` file.
- The test reports accurate coverage results based on the loaded 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` 2ms 3

AI ASSESSMENT

Scenario: The test verifies that the `load_metadata_file` function correctly loads metadata from a PyProject file.

Why Needed: This test prevents a bug where the function fails to load metadata from a non-existent or corrupted PyProject file.

Key Assertions:

- pyproject.toml should be created at the specified path.
- The `load_metadata_file` function should return an empty list if no metadata is found.
- The `load_metadata_file` function should raise a `FileNotFoundError` when trying to load metadata from a non-existent file.
- The `load_metadata_file` function should raise a `NotADirectoryError` when trying to load metadata from a directory that does not contain a PyProject file.
- The `load_metadata_file` function should return an empty list if the file is corrupted or invalid.
- The `load_metadata_file` function should correctly handle cases where the file is missing a required section (e.g., `[tool.scripts]`)
- The `load_metadata_file` function should not raise any exceptions when trying to load metadata from a non-existent PyProject file with an empty directory 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/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` configuration is loaded correctly from the PyProject.toml file.

Why Needed: This test prevents a potential bug where the `ollama_host` configuration is not loaded, potentially leading to incorrect behavior or errors in downstream applications.

Key Assertions:

- The contents of the `pyproject.toml` file are correct and match the expected values.
- The `ollama_host` configuration is present and has the correct value.
- The `ollama_host` configuration is correctly formatted as a string.
- The `pyproject.toml` file is written to disk with the correct permissions.
- The `pyproject.toml` file is not empty or corrupted.
- The `ollama_host` configuration does not contain any invalid characters or syntax.
- The `ollama_host` configuration is correctly formatted as a string in the expected format.
- The `pyproject.toml` file is written to disk with the correct timestamp.

COVERAGE

src/pytest_llm_report/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: Verify that the `load_omit_tests_from_coverage` function can load omit_tests_from_coverage configuration from a PyProject file.

Why Needed: This test prevents regression where the `load_omit_tests_from_coverage` function fails to load omit_tests_from_coverage configuration from a PyProject file.

Key Assertions:

- The `pyproject.toml` file is successfully read and parsed by the `load_omit_tests_from_coverage` function.
- The `load_omit_tests_from_coverage` function correctly loads the `omit_tests_from_coverage` section from the PyProject file.
- The `load_omit_tests_from_coverage` function does not raise an exception when encountering a missing or invalid `omit_tests_from_coverage` section in the PyProject file.
- The `load_omit_tests_from_coverage` function preserves the original configuration values for other sections in the PyProject file.
- The `load_omit_tests_from_coverage` function correctly handles nested configurations and sub-sections.
- The `load_omit_tests_from_coverage` function does not modify any existing configuration settings in the PyProject file.
- The `load_omit_tests_from_coverage` function preserves the original indentation and line breaks in the PyProject 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, 362-364, 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 correctly, potentially causing coverage issues or incorrect results.

Key Assertions:

- The `pyproject.toml` file contains the correct value for `param_value_max_chars`.
- The `param_value_max_chars` value in the PyProject.toml file matches the expected value.
- The `load_param_value_max_chars()` function correctly loads the `param_value_max_chars` value from the PyProject.toml file.
- The coverage of the test is not affected by the incorrect loading of `param_value_max_chars`.
- The `pyproject.toml` file is properly formatted and contains the correct values for all parameters.
- The `load_param_value_max_chars()` function correctly handles invalid or missing values 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	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

Scenario: The test verifies that the 'collect' option is only loaded when a specific configuration file (pyproject.toml) exists.

Why Needed: This test prevents a potential bug where the 'collect' option is accidentally loaded from other files, potentially causing unexpected behavior or errors.

Key Assertions:

- The 'collect' option should be present in pyproject.toml and not in any other file.
- The 'collect' option should only be written to pyproject.toml if it exists.
- Any changes made to the 'collect' option should be reflected in pyproject.toml.
- No changes to the 'collect' option should be made from other files.
- The test should fail when a non-existent file (e.g., .gitignore) is used instead of pyproject.toml.
- The test should pass when the correct configuration file exists and the 'collect' option is present in 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	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 and set to the specified seconds.

Why Needed: This test prevents a potential bug where the timeout value is not properly loaded or set, potentially leading to incorrect behavior when loading files with timeouts.

Key Assertions:

- The value of `timeout_seconds` in `pyproject.toml` is correctly read from the file and matches the specified seconds.
- The value of `timeout_seconds` in `pyproject.toml` is set to the correct number of seconds (e.g., 30, 60, etc.) when loading a project.
- The test loads files with timeouts using the expected timeout value.
- The test verifies that loading files with timeouts does not cause any errors or unexpected behavior.
- The test checks that the `timeout_seconds` value is correctly overridden by environment variables or other settings if it is not specified in `pyproject.toml`.
- The test ensures that the `timeout_seconds` value is not accidentally set to a lower value than expected (e.g., 1, 2, etc.)
- The test verifies that loading files with timeouts does not cause any unexpected behavior or errors due to incorrect timeout 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	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: Tests the configuration of aggregation settings.

Why Needed: Prevents a potential bug where the aggregate directory is not set correctly, leading to incorrect reporting.

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: Tests Config with all output paths.

Why Needed: Prevents a potential bug where the report configuration is not set correctly, causing inconsistent output files.

Key Assertions:

- The `report_html` attribute of the config object should be equal to 'report.html'.
- The `report_json` attribute of the config object should be equal to 'report.json'.
- The `report_pdf` attribute of the config object should be equal 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` in the test configuration.

Why Needed: This test prevents a potential issue where the test fails due to an incorrect capture settings.

Key Assertions:

- The `capture_failed_output` attribute of the test configuration is set to `True`.
- The maximum number of characters for capturing output is set to `8000`.
- The captured output will not exceed `8000` characters.
- The test will fail if `capture_failed_output` is set to `False` or a non-integer value.
- The capture settings are consistent across all tests in this module.
- The maximum number of characters for capturing output is sufficient for most use cases.
- The captured output will not exceed the specified limit.

COVERAGE

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

Why Needed: Prevents a potential bug where the configuration file is not set correctly, potentially leading to incorrect metadata or HMAC key usage.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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: Test the configuration of coverage settings.

Why Needed: Prevents a bug where coverage settings are not correctly applied to all tests.

Key Assertions:

- config.omit_tests_from_coverage is set to False which means that all tests will be included in the coverage report.
- config.include_phase == 'all' ensures that all test phases are covered by the configuration.
- The assertion checks if config omits tests from coverage and includes phase correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Test the ability to include custom exclude globs in the LLM context configuration.

Why Needed: This test prevents a potential bug where the default exclude globs are not properly included in the LLM context configuration when custom exclude globs are provided.

Key Assertions:

- The '*.pyc' and '*.log' exclude globs should be present in the config.llm_context_exclude_globs list.
- The '*.class' exclude glob should also be included in the list if it is a valid Python file name.
- If no custom exclude globs are provided, the default exclude globs should still be included in the config.llm_context_exclude_globs list.
- The order of the exclude globs in the config.llm_context_exclude_globs list should match the order they appear in the test configuration.
- The '*.class' exclude glob should not be present in the config.llm_context_exclude_globs list if it is a valid Python file name.
- If the custom exclude glob '*.*' is provided, it should include all files and directories.
- The custom exclude glob '**/*.pyc' should also include all files and directories that match the pattern '*.pyc'.
- The custom exclude glob '**/*.log' should also include all files and directories that match the pattern '*.log'.

COVERAGE

src/pytest_llm_report/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 `llm_context_include_globs` configuration option includes only `.py` files.

Why Needed: Prevent a potential bug where the include globs are not correctly configured, potentially leading to incorrect or missing file inclusion.

Key Assertions:

- The `*.py` glob matches only files with a `.py` extension.
- The `*.pyi` glob matches only files with a `.pyi` extension.
- The `llm_context_include_globs` configuration option is set to include both `.py` and `.pyi` files.
- The `*.py` file does not have a `.pyi` extension.
- The `*.pyi` file does not have a `.py` extension.
- The `include_globs` configuration option is correctly applied to the LLM context.
- The test fails when the include globs are not correctly configured, indicating a bug in the configuration.

COVERAGE

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

Scenario: Verify that `include_pytest_invocation` is set to False for the specified configuration.

Why Needed: This test prevents a potential bug where `include_pytest_invocation` is accidentally set to True, potentially causing issues with pytest invocation settings.

Key Assertions:

- config.include_pytest_invocation
- is not equal to Config(include_pytest_invocation=True)
- includes --api-key=\S+ in the invocation redact patterns
- does not contain --api-key=\S+ in the invocation redact patterns
- includes --pytest-arg-parsing-redact in the invocation redact patterns
- does not contain --pytest-arg-parsing-redact in the invocation redact patterns
- is set to False for the specified configuration
- includes pytest in the invocation settings

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 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: Verifies that the LLM execution settings are correctly configured.

Why Needed: Prevents a potential bug where the maximum number of tests is not set to 50, which could lead to performance issues or unexpected behavior in the test suite.

Key Assertions:

- assert config.llm_max_tests == 50
- assert config.llm_max_concurrency == 8
- assert config.llm_requests_per_minute == 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: Test the LLM parameter settings configuration.

Why Needed: Prevents a potential bug where the maximum length of LLM parameter values is not checked correctly.

Key Assertions:

- The `llm_include_param_values` attribute of the Config object is set to True.
- The value of `llm_param_value_max_chars` in the Config object is set to 200.
- The `llm_param_value_max_chars` attribute of the Config object is not equal to 200.
- The `llm_include_param_values` attribute of the Config object is not set to True.
- The value of `llm_param_value_max_chars` in the Config object is less than or equal to 0.
- The `llm_param_value_max_chars` attribute of the Config object is greater than 200.
- The `llm_include_param_values` attribute of the Config object is set to False.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Tests the configuration of LLM settings for OLLAMA provider.

Why Needed: Prevents a potential bug where the provided LLM context bytes exceeds the limit.

Key Assertions:

- The `provider` attribute is set to 'ollama'.
- The `model` attribute is set to 'llama3.2'.
- The value of `llm_context_bytes` is equal to 64000.
- The value of `llm_context_file_limit` is less than or equal to 20.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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 Config object is set to the expected value.

Why Needed: This test prevents a potential bug where the `repo_root` attribute is not correctly initialized when using a relative path.

Key Assertions:

- The `repo_root` attribute should be equal to the provided `Path` object.
- The `repo_root` attribute should be an instance of `path.Path`.
- The `repo_root` attribute should have a string value that is equal to the provided `Path` object.
- The `repo_root` attribute should not be set to a relative path (e.g., `/project/...`).
- The `repo_root` attribute should not be set to an absolute path (e.g., `/home/user/project/...`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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 `Config` class with valid phase values.

Why Needed: Prevents a potential bug where invalid include_phase values cause validation to fail.

Key Assertions:

- The `validate()` method of the `Config` class should not return any errors for all valid include_phase values.
- All included phases (run, setup, teardown, and all) should pass validation without any issues.
- No error messages or warnings should be displayed when validating a valid configuration.
- The `include_phase` attribute is correctly set to the specified phase value for each test case.
- The `validate()` method does not return an empty list of errors even if none are present.
- Any invalid include_phase values should be ignored and not cause validation to fail.
- The `Config` class properly handles cases where include_phase values are missing or set to an unknown value.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`20 lines (ranges: 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: Test the default exclude globs for the llm context.

Why Needed: This test prevents a potential bug where the default exclude globs are not correctly identified, potentially leading to incorrect configuration of the LLM.

Key Assertions:

- The function `Config().llm_context_exclude_globs` returns a list containing the specified globs.
- * The glob pattern `*.pyc` is included in the returned list.
- * The glob pattern `__pycache__/*` is included in the returned list.
- * The glob pattern `*secret*` is included in the returned list.
- * The glob pattern `*password*` is included in the returned 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)

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_redact_patterns` 1ms 3

AI ASSESSMENT

Scenario: Verify that the default redact patterns include sensitive information.

Why Needed: This test prevents a potential security vulnerability where sensitive information is not properly redacted.

Key Assertions:

- Any pattern containing '--password' should be included in the default redact patterns.
- Any pattern containing '--token' should be included in the default redact patterns.
- Any pattern containing '--api[_-]?key' should be included 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: Verifies that the default configuration values are correctly set.

Why Needed: This test prevents a potential bug where the default configuration values are not properly set, potentially leading to incorrect behavior in downstream tests or code.

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: Test that the `is_llm_enabled` method returns correct enabled status for different providers.

Why Needed: This test prevents a regression where the `is_llm_enabled` method might return incorrect results due to changes in provider settings.

Key Assertions:

- The function should return False when the provider is set to 'none'.
- The function should return True when the provider is set to 'ollama' or '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	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

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_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 validates an invalid include phase.

Why Needed: Prevents a potential bug where the test does not catch and report an error for an invalid include phase.

Key Assertions:

- The test should validate that there is exactly one error message.
- The error message should contain 'Invalid include_phase 'invalid''.
- The error message should be present in the first error object.

COVERAGE

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

Scenario: Test validates an invalid provider.

Why Needed: Prevents a potential bug where the test does not handle an invalid provider correctly.

Key Assertions:

- The `validate()` method returns at least one error for the given invalid provider.
- The error message contains the string 'Invalid provider 'invalid' to identify the issue.
- The test asserts that there is exactly one error related to the invalid provider.

COVERAGE

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

Why Needed: This test prevents regression where the Config class does not validate numeric bounds correctly.

Key Assertions:

- The config should contain errors for invalid numeric values.
- The config should include 'llm_context_bytes' in its error messages.
- The config should include 'llm_max_tests' in its error messages.
- The config should include 'llm_requests_per_minute' in its error messages.
- The config should include 'llm_timeout_seconds' in its error messages.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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 a Config object with a valid configuration.

Why Needed: Prevents a potential bug where an invalid configuration is passed to the validate method, leading to incorrect results or errors.

Key Assertions:

- The config should return an empty list when validated.
- The validate method should not throw any exceptions for a valid configuration.
- No ValueError should be raised when validating a Config object with no issues.
- The config's validate method should only return an empty list for valid configurations.
- The validate method should not raise an exception if the input is invalid but the config is still considered valid.
- A Config object with a valid configuration should have its validate method return an empty list.

COVERAGE

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

Key Assertions:

- The `cfg` variable should be an instance of the `Config` class.
- The `cfg` variable should not have any registered options.
- The `cfg` variable should have safe defaults (e.g. no sensitive information).
- The `cfg` variable's type should be consistent with the expected default configuration.
- No exceptions should be raised when trying to create an instance of `Config` from the given pytestconfig.
- The config's options should not be registered or modified during its creation.
- The config's values should not contain sensitive information (e.g. passwords, API 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/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.

Key Assertions:

- The `pytestconfig` object should be accessible.
- The `pytestconfig` object should not be None.
- The `pytestconfig` object should have attributes that can be accessed (e.g., `markers`, `options`, etc.).

COVERAGE

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

PASSED `tests/test_plugin_integration.py::TestPluginHooksWithPytester::test_both_json_and_html_outputs` 92ms 8

AI ASSESSMENT

Scenario: Test generates both JSON and HTML reports for a test function.

Why Needed: Prevents regression in case of changes to the test function or its dependencies.

Key Assertions:

- The test function `test_simple()` should be executed without any errors.
- The generated JSON report should contain the expected output.
- The generated HTML report should also contain the expected output.
- The existence of both JSON and HTML reports should be verified.
- The directories for JSON and HTML reports should exist at the specified paths.
- The test function `test_simple()` should not raise any exceptions during execution.

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` 64ms 7

AI ASSESSMENT

Scenario: Test that the `test_collection_finish_counts_items` test verifies the correct number of items collected by pytest_collection_finish.

Why Needed: This test prevents a regression where the plugin incorrectly reports an incorrect number of items collected.

Key Assertions:

- The 'collected_count' key in the report.json file should contain the total number of items collected (3) as verified by the `pytest_collection_finish` hook.
- The 'collected_count' key in the report.json file should be an integer value (3).
- The 'collected_count' key in the report.json file should match the expected count before running pytest_collection_finish (3).

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` 63ms 7

AI ASSESSMENT

Scenario: Test that output directories are created if missing.

Why Needed: Prevents a potential issue where the test fails due to missing output directories.

Key Assertions:

- The `nested` directory should be created in the report.json file.
- The `report.json` file should exist in the specified location.
- The `pytester.path` attribute should have been updated with the correct path to the report.json file.
- The `pytester.runpytest` method should have executed successfully without raising an exception.
- The `assert` statement inside the test function should evaluate to True.

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` 187ms 7

AI ASSESSMENT

Scenario: Test that fixture errors are captured in report.

Why Needed: Prevents a regression where the test fails with an error message instead of a meaningful exception.

Key Assertions:

- The 'report.json' file is created and contains an error count of 1.
- The error message in the 'report.json' file indicates that a fixture failed.
- The assertion `data['summary']['error'] == 1` checks if the error count is indeed 1.

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` 179ms 7

AI ASSESSMENT

Scenario: Test pytest_runtest_makereport captures all outcomes, ensuring that the report includes both passed and failed tests as well as skipped tests.

Why Needed: This test prevents a regression where pytest_runtest_makereport might not capture all types of test outcomes (e.g., skipped tests) and instead only reports passed or failed tests.

Key Assertions:

- The report should include both 'passed' and 'failed' test outcomes, as well as 'skipped' test outcomes.
- The report should include the names of all tests that were skipped.
- The report should not exclude any test outcomes (e.g., it should report 'skipped' if a test is skipped).
- The report should include all test outcomes, regardless of whether they are passed or failed.
- The report should be able to distinguish between passed and failed tests based on the test function's return value.
- The report should not incorrectly report any test outcomes (e.g., it should not report a skipped test as 'passed' if it was actually failed).
- The report should include all types of test outcomes, including skipped tests.

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` 58ms 3

AI ASSESSMENT

Scenario: Verify that no report is generated when the output path is disabled.

Why Needed: This test prevents a regression where reports are not generated even though output paths are specified.

Key Assertions:

- The `report_path` should exist after running `pytester.runpytest()` without any output path specified.
- No error message should be displayed when attempting to create the report file.
- The file 'report.json' should not be created in the current working directory.
- The file 'report.json' should not be created in any subdirectories of the current working directory.
- The `pytester.path` attribute should still exist after running `pytester.runpytest()` without an output path specified.
- No exception should be raised when attempting to create the report file.
- The `report_path` should be a valid file system 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	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 625ms 8

AI ASSESSMENT

Scenario: Test that --llm-pdf option enables the plugin.

Why Needed: Prevents regression where the plugin is not enabled due to a missing Playwright configuration.

Key Assertions:

- The `test_pass()` function should be executed without any errors or warnings.
- The `pytester.runpytest` command with `--llm-pdf=report.pdf` should exit with code 0 (success) or warning, but definitely run.
- If the plugin wasn't enabled, it wouldn't try and fail due to missing playwright configuration.
- The `test_pass()` function should be executed without any errors or warnings if only `--llm-pdf` is used.

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` 66ms 7

AI ASSESSMENT

Scenario: Verify that the pytest_sessionstart records start time is captured correctly.

Why Needed: This test prevents a potential bug where the start time of the session is not recorded correctly, potentially leading to incorrect reporting or analysis.

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 test execution.
- The pytest_sessionstart records should include a timestamp that corresponds to the start time of the session.

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

Scenario: Test the functionality of LLM context marker in a test.

Why Needed: This test prevents regression and ensures that the LLM context marker does not cause errors during testing.

Key Assertions:

- The test verifies whether the context marker is correctly applied to the test.
- The test checks if any errors are thrown due to the context marker.
- The test verifies that the context marker is properly removed after the test completes.
- The test ensures that the context marker does not interfere with other tests or plugins.
- The test checks for any unexpected behavior or side effects of the context marker.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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` function does not throw any exceptions when called.

Why Needed: This test prevents a potential bug where the `requirement_marker` function could be used to silently ignore or mask errors in other parts of the codebase.

Key Assertions:

- The `requirement_marker` function should not raise an exception.
- The `requirement_marker` function should not modify any external state that is critical to the test's assertions.
- The `requirement_marker` function should only return a boolean value indicating whether the requirement was met or not.
- The `requirement_marker` function should not throw any exceptions, even if an error occurs during execution.
- The `requirement_marker` function should not modify any external state that is critical to the test's assertions, except for the case where no requirements are met and the function returns False.
- The `requirement_marker` function should only return a boolean value indicating whether the requirement was met or not, without modifying any external state.

COVERAGE

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

PASSED `tests/test_plugin_integration.py::TestReportGeneration::test_report_writer_integration` 38ms 6

AI ASSESSMENT

Scenario: Test the integration of report writer with pytest_llm_report module.

Why Needed: This test prevents a potential regression where the report writer does not generate a full report even when there are multiple tests that pass or fail.

Key Assertions:

- Verify that the report writer generates a JSON file containing the total number of tests and passed tests.
- Verify that the HTML file contains references to all test files (test_a.py and test_b.py).
- Assert that the 'report.json' file exists in the temporary directory.
- Load the JSON data from the 'report.json' file and verify its structure.
- Check if the total number of tests is 2 and the number of passed tests is 1 in the JSON data.
- Verify that the HTML file contains references to all test files (test_a.py and test_b.py) 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	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` 3ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that collectreport skips when disabled and pytest_collectreport is mocked correctly.

Why Needed: To prevent pytest_collectreport from being executed when the plugin's collect report is disabled.

Key Assertions:

- The `pytest_collectreport` function should be called with `_enabled_key` set to `False`.
- The `get` method of the stash session should return `None` when `_enabled_key` is `False`.
- The `session.config.stash.get` method should not call `pytest_collectreport` or any other functions related to collect report.
- The `pytest_collectreport` function should be mocked with a `MagicMock` object that returns a `True` value for the stash session configuration.
- The `get` method of the stash session should return `None` when `_enabled_key` is set to `False` and the stash session configuration is `True`.
- The `pytest_collectreport` function should not be called with any arguments when the stash session configuration 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	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: Test that collectreport calls collector when enable is True.

Why Needed: This test prevents a potential regression where the plugin does not call the collector even when collectreport is enabled.

Key Assertions:

- The `pytest_collectreport` function should be called with the mock report object.
- The `handle_collection_report` method of the mock collector should be called once with the mock report.
- The `stash_get` function should return True for the `_enabled_key` and `_collector_key` keys when the collectreport 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`	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 the `session` attribute is missing from a report.

Why Needed: Prevent regression in plugin behavior when using Pytest without a session.

Key Assertions:

- The function should not raise an error or any exception.
- The `session` attribute of the mock report object should be deleted before calling `pytest_collectreport()`.
- The `pytest_collectreport()` function should not throw an exception when called with a mock report object that does not have a `session` attribute.
- The `pytest_collectreport()` function should behave as expected without raising any exceptions or errors.
- The `pytest_collectreport()` function should be able to handle the absence of the `session` attribute in the 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`	`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: The test verifies that the `pytest_collectreport` function does not raise an error when a `session` is set to `None`.

Why Needed: To prevent a potential bug where the plugin fails to collect reports due to a missing session.

Key Assertions:

- mock_report.session is None
- pytest_collectreport(mock_report) should not be called with an invalid argument (session)
- The `session` attribute of `mock_report` is still set to `None` after the test finishes

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/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 regression where the LLM enabled warning is not raised correctly.

Key Assertions:

- The `pytest_llm_report` plugin writes to the `pyproject.toml` file with the correct configuration.
- The `LLM` setting in the `pyproject.toml` file is set to `True` when LLM is enabled.
- The `pytest_llm_report` plugin raises a warning in the `pyproject.toml` file with the correct message.
- The warning message includes the expected error message for LLM enabled configuration.
- The warning message includes the expected error code for LLM enabled configuration.
- The warning message is raised before any tests are run.
- The warning message does not occur when LLM is disabled in the `pyproject.toml` file.
- The warning message is only raised if LLM is enabled 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	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: The test verifies that the `pytest_configure` function raises a `UsageError` when given an invalid or malformed `pyproject.toml` file.

Why Needed: This test prevents a potential bug where the plugin incorrectly handles configuration errors and raises a generic `UsageError` instead of providing meaningful feedback.

Key Assertions:

- The `pytest_configure` function should raise a `UsageError` when given an invalid or malformed `pyproject.toml` file.
- The error message raised by `pytest_configure` should be informative and specific to the issue (e.g. 'Invalid configuration: ...')
- The test should fail with a meaningful error message indicating that the configuration is invalid
- The plugin's behavior should change when given an invalid or malformed `pyproject.toml` file, e.g. by displaying a more detailed error message or providing additional feedback

COVERAGE

src/pytest_llm_report/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 and ensures addinivalue_line is not called for markers before worker check.

Why Needed: This test prevents a potential regression where the plugin might skip configuration due to an incorrect assumption about the number of workers, potentially leading to issues with marker handling.

Key Assertions:

- mock_config.addinivalue_line was called but addinivalue_line is still called for markers before worker check
- mock_config.workerinput was set correctly

COVERAGE

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

Scenario: Test that fallback to load_config is triggered when Config.load is missing.

Why Needed: This test prevents a potential bug where the plugin does not load configurations due to missing Config.load method.

Key Assertions:

- The Config.load method is called with no arguments.
- The mock_load function is called once with mock_cfg as argument.
- mock_cfg.validate() returns an empty list.
- The mock_load function returns mock_cfg.
- pytest_configure(mock_config) is called.
- The mock_load function does not call any other functions or methods.
- The Config.load method is not called in the test.
- The pytest_llm_report.options.Config class has a no-arg __init__ method.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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 the CLI options can override the pyproject.toml configuration.

Why Needed: This test prevents a potential regression where CLI options are not overriding pyproject.toml configuration, potentially causing issues with plugin loading.

Key Assertions:

- pyproject.toml is created and written to disk successfully.
- CLI options can override pyproject.toml configuration.
- Plugin load configuration overrides pyproject.toml configuration.
- CLI options take precedence over pyproject.toml configuration for plugins.
- Pytest LLM report options module 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	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` 2ms 3

AI ASSESSMENT

Scenario: The test verifies that the plugin can successfully load configuration options from a PyProject.toml file.

Why Needed: This test prevents potential issues where plugins are unable to access or read configuration files.

Key Assertions:

- pyproject.toml is created with the correct contents and path.
- The plugin can write to pyproject.toml without errors.
- The plugin can load configuration options from pyproject.toml without errors.
- The plugin's configuration file is not empty or contains invalid data.
- The plugin does not throw any exceptions when writing to pyproject.toml.
- The plugin does not throw any exceptions when loading configuration options from pyproject.toml.
- The plugin uses the correct path for pyproject.toml.

COVERAGE

src/pytest_llm_report/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 is not properly initialized or configured.

Key Assertions:

- The `pytest_terminal_summary` function does not check if the plugin is enabled before calling `stash.get()` on it.
- The `stash.get()` method should be called with a valid key and value to retrieve the stash data.
- The `stash.get()` method should raise an exception or return an error when the stash data 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/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 and returns early without doing anything.

Why Needed: This test prevents a regression where the plugin might not skip the terminal summary on xdist workers due to incorrect configuration or unexpected behavior.

Key Assertions:

- mock_config.workerinput is set to 'gw0'
- result is None (indicating early return without doing anything)
- mocked pytest_terminal_summary function does not perform any actions

COVERAGE

src/pytest_llm_report/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: To prevent a regression where the test fails due to makereport being disabled.

Key Assertions:

- mock_item.config.stash.get returns False
- mock_call.return_value is None
- mock_outcome.get_result.return_value is None
- gen.send(mock_outcome) raises StopIteration

COVERAGE

`src/pytest_llm_report/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 makereport calls collector when enabled.

Why Needed: Prevents test from failing due to a missing or incorrect implementation of pytest_runtest_makereport.

Key Assertions:

- The function `pytest_runtest_makereport` is called with the correct mock item and report.
- The mock collector is properly configured to handle runtest log reports.
- The mock report is returned from the stash_get method of the mock item.
- The mock collector handles the runtest log report correctly.
- No StopIteration exception is raised during the execution of the test.
- The mock collector logs the runtest log report with the correct mock report.
- The mock collector calls the `handle_runtest_logreport` method on the mock item with the correct mock report.
- The stash_get method returns True for the _enabled_key and _collector_key keys when called.

COVERAGE

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

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_disabled` 2ms 2

AI ASSESSMENT

Scenario: Test that collection_finish is skipped when disabled and pytest_collection_finish is called.

Why Needed: This test prevents a regression where collection_finish is not properly handled when pytest_collection_finish is disabled.

Key Assertions:

- mock_session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_collection_finish(mock_session) should be called with the correct arguments
- mock_session.config.stash.get.return_value should return False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Verify that collection_finish is called when collection finish is enabled.

Why Needed: This test prevents a bug where the collector is not called when collection finish is enabled, potentially leading to incorrect data being collected.

Key Assertions:

- The stash_get method returns True for _enabled_key and mock_collector.
- The stash_get method does not return default value for _collector_key.
- mock_collector.handle_collection_finish is called once with the correct arguments.
- The collection finish data is collected 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	10 lines (ranges: 363-364, 367, 371-373, 407, 411-413)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_disabled` 2ms 2

AI ASSESSMENT

Scenario: Test that sessionstart skips when disabled and checks enabled status.

Why Needed: Prevents a potential bug where the plugin does not properly handle pytest session start when disabled.

Key Assertions:

- mock_session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_sessionstart(mock_session) should be called with _enabled_key set to False
- mock_session.config.stash.get.return_value should be set to False
- pytest_sessionstart(mock_session) should not have checked enabled status

COVERAGE

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

Why Needed: Prevents a potential bug where the collector is not initialized due to an uninitialized stash.

Key Assertions:

- The key '_enabled_key' should be present in the mock_stash dictionary.
- _collector_key should be present in the mock_stash dictionary.
- The value of _start_time_key should be present in the mock_stash dictionary.
- The collector should be created with a properly initialized stash.
- The stash should support both get() and [] operations without raising an error.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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` 4ms 2

AI ASSESSMENT

Scenario: Test pytest_addoption adds expected arguments to the parser.

Why Needed: This test prevents a potential bug where pytest_addoption does not add all required arguments to the parser, potentially leading to unexpected behavior or errors.

Key Assertions:

- parser.getgroup.assert_called_with('llm-report', 'LLM-enhanced test reports')
- group.addoption.call_args_list[0][0] == '--llm-report'
- group.addoption.call_args_list[1][0] == '--llm-coverage-source'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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 that pytest_addoption no longer adds INI options when run without an .ini file.

Why Needed: This test prevents a regression where the plugin incorrectly added INI options to the command line arguments.

Key Assertions:

- parser.addini was not called.
- parser.addini was not called with any arguments.
- parser.addini was not called with an option that requires an .ini file (e.g., -i, --ini).
- parser.addini was not called with a valid .ini file path.
- parser.addini was not called with a valid .ini file name.
- parser.addini was not called with any command line arguments.
- parser.addini was not called with an option that requires an .ini file (e.g., -i, --ini).
- parser.addini was not called with a valid .ini file path.
- parser.addini was not called with a valid .ini file name.
- parser.addini was not called with any command line arguments.
- parser.addini was not called with an option that requires an .ini file (e.g., -i, --ini).
- parser.addini was not 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/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 that terminal summary with LLM enabled runs correctly and passes the required configuration.

Why Needed: This test prevents regression in the plugin's functionality when LLM is enabled.

Key Assertions:

- The `pytest_terminal_summary_llm_enabled` function should be called with a valid configuration.
- The `cfg` variable passed to `pytest_terminal_summary_llm_enabled` should have `_enabled_key` and `_config_key` set correctly.
- The `mock_annotate.call_args[0][1] == cfg` assertion should pass when the correct configuration is provided.
- The `mock_annotate.call_args[0][1]` variable should contain the correct value from the `cfg` dictionary.
- The `pytest_terminal_summary_llm_enabled` function should not raise any exceptions even if an error occurs during execution.
- The `pytest_terminal_summary_llm_enabled` function should correctly handle the case when the LLM is disabled.
- The `pytest_terminal_summary_llm_enabled` function should not fail or produce unexpected results when run with a valid configuration but without LLM enabled.
- The `pytest_terminal_summary_llm_enabled` function should correctly report the terminal summary even if the LLM is disabled.

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: Prevents regression where terminal summary does not include a collector even when it's required.

Key Assertions:

- The `pytest_terminal_summary` function should be able to create a stash that supports both get() and [] indexing.
- The `_enabled_key` and `_config_key` should match the keys used in the mock stash.
- The `stash` dictionary returned by the mock stash should contain the expected values for `_enabled_key` and `_config_key`.
- The `mock_config.stash` attribute should be set to the mock stash created using the `MockStash` class.
- The `pytest_terminal_summary` function should not raise an error when given a mock config that does not have both `_enabled_key` and `_config_key` keys.
- The `mock_writer_cls.return_value` attribute should return a mock writer object with the correct `ReportWriter` instance.
- The `mock_mapper_cls.return_value` attribute should return a mock mapper object with the correct `CoverageMapper` instance.
- The `mock_mapper.map_coverage.return_value` attribute should be an empty dictionary when called.

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` 3ms 3

AI ASSESSMENT

Scenario: Test terminal summary with aggregation enabled.

Why Needed: This test prevents a regression where the plugin does not aggregate terminal summaries correctly when `aggregate_dir` is set to `/agg`.

Key Assertions:

- The stash should support both get() and [] operations.
- The aggregator should be called once with the correct arguments.
- The report writer should write JSON and HTML files successfully.
- The aggregation result should be returned by the Aggregator instance.
- The aggregate directory should be set to `/agg` for the test to pass.
- The stash should have `_enabled_key` and `_config_key` attributes with the correct values.
- The mock terminal reporter should call `aggregate` on the aggregator 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	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` 5ms 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	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 with a test file and dependency file.

Why Needed: This test prevents regression by ensuring the context assembler correctly identifies dependencies for a balanced context.

Key Assertions:

- The 'utils.py' file is present in the assembled context.
- The 'def util()' function is found in the 'utils.py' file within the assembled context.
- The dependency on 'util()' from 'test_a.py' is correctly identified and included in the assembled context.

COVERAGE

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

Why Needed: This test prevents regression when using the 'complete' llm_context_mode, as it ensures that the assembler can correctly identify and include all necessary source files in the assembled context.

Key Assertions:

- The assembler should be able to assemble a complete context for the specified test file.
- The assembler should include all necessary source files in the assembled context.
- The assembler should not include any unnecessary source files in the assembled context.
- The assembler should correctly identify and include the specified function in the assembled context.
- The assembler should raise an exception if it encounters a required source file that is missing from the test file.
- The assembler should handle cases where the test file has multiple functions with the same name but different bodies.
- The assembler should preserve the original directory structure of the test file when assembling 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/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: Test the ContextAssembler with minimal context mode to ensure it correctly assembles a test file and returns an empty context.

Why Needed: This test prevents regression when using minimal context mode, ensuring that the assembler can still assemble a test file without modifying its source code.

Key Assertions:

- The 'test_1' function is present in the assembled source code.
- The context of the assembled source code is empty.
- The assembler returns an empty context when assembling a test file with minimal context mode.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: Test the ContextAssembler with balanced context limits to ensure it handles large files correctly.

Why Needed: This test prevents a potential bug where the ContextAssembler exceeds the specified context limit when assembling a file that is too long, causing unnecessary truncation of the content.

Key Assertions:

- The 'f1.py' file in the context is truncated to 40 bytes or less.
- The length of the 'f1.py' file in the context does not exceed 60 bytes (20 bytes + truncation message).
- The ContextAssembler correctly handles large files by only including a portion of the content up to 40 bytes.
- The ContextAssembler prevents unnecessary truncation of long files by limiting the amount of content included in the context.
- The test result indicates that the ContextAssembler successfully assembled the file without excessive truncation.
- The test result shows that the ContextAssembler correctly handles large files and does not include unnecessary content.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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: Verify the ContextAssembler correctly handles non-existent files and nested test names with parameters.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly returns an empty string for a file that does not exist or has a nested test name with parameters.

Key Assertions:

- The function `_get_test_source` correctly returns an empty string when given a non-existent file.
- The function `_get_test_source` correctly handles nested test names with parameters by including the parameter in the source code.
- The function `_get_test_source` raises an AssertionError for a valid test name with parameters.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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` 2ms 4

AI ASSESSMENT

Scenario: The test verifies that the ContextAssembler excludes certain files from the LLM context.

Why Needed: This test prevents a potential bug where sensitive files are included in the LLM context.

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 in minimal mode returns no context files.

Why Needed: Prevents regression where the assembler does not generate any context files when run in minimal mode.

Key Assertions:

- The `context_files` attribute of the `assemble` method is set to an empty list.
- The `test_source` variable contains the original test code.
- The `context_files` attribute is checked against this value after assembly.

COVERAGE

src/pytest_llm_report/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` 2ms 4

AI ASSESSMENT

Scenario: Test assemble respects llm_context_override from test.

Why Needed: Prevents regression when using ContextAssembler with overridden LLM context.

Key Assertions:

- The assembler should use the specified 'balanced' mode for LLM context override.
- The module file 'module.py' should be included in the context files.
- The coverage entry should include the 'module.py' file and line ranges of '1'.
- The coverage entry should have a line count of 1.
- The assembler should respect the specified LLM context override ('balanced') even when using minimal 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	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

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: 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 the scenario where a balanced context is assembled without a file.

Why Needed: This test prevents a regression that occurs when a non-existent file is encountered in the balanced context.

Key Assertions:

- The `ContextAssembler` returns an empty dictionary for the specified test file.
- The coverage entry for the specified test file does not include any lines.
- The test file exists at the expected location (nonexistent.py).
- The test file is not skipped by the balanced context due to its existence.
- The coverage report includes all lines from the non-existent file.
- The `ContextAssembler` correctly returns an empty dictionary for a non-existent file.

COVERAGE

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

Key Assertions:

- The content of the test module is truncated to 120 bytes or less.
- A message indicating that the content was truncated is present in the test module.
- The total size of the content exceeds the limit when using the balanced context.
- The total size of the content exceeds the limit when using the balanced context and a file limit.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 potential bug where the test_foo function is not covered by the code under test, leading to incorrect results.

Key Assertions:

- The `ContextAssembler` returns an empty dictionary when there are no lines of code in the test file.
- The `TestContextAssemblerEdgeCases.test_balanced_context_no_coverage` test verifies that this happens.
- When running tests with no coverage, the `ContextAssembler` should return an empty dictionary.
- The expected result is an empty dictionary, indicating no coverage.
- The absence of code in the test file affects the coverage calculation.
- Without any lines of code in the test file, there are no nodes to assemble and cover.
- The `_get_balanced_context` method should return an empty dictionary when no coverage is present.
- When `test_foo` is not covered by the code under test, the context should be empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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: The test verifies that the ContextAssembler's balanced context does not exceed the maximum bytes before processing a file.

Why Needed: This test prevents a regression where the ContextAssembler might incorrectly report an error or fail to assemble a balanced context when it exceeds the specified limit of bytes.

Key Assertions:

- The length of the assembled context should be either 0 (if no files were processed) or 1 (if only one file was processed).
- If two files are processed, the assembled context should have exactly one element (i.e., neither file is fully 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	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 regression where incomplete contexts are used with unbalanced configurations.

Key Assertions:

- The `context` variable should contain the same file path as the input module.
- The number of lines in the coverage entry should match the line count in the input module.
- The file path 'module.py' should be present in the `context` 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/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 assembler to fail or return unexpected results.

Key Assertions:

- The function `_get_test_source` is called with an empty string as the first argument.
- The function `_get_test_source` returns an empty string as its result.
- The `split` method of the empty string is called, which should not be necessary and may cause unexpected behavior.
- The `tmp_path` parameter passed to `_get_test_source` is not being used or is being modified in some way that affects the test outcome.
- The `assert` statement checks for an equality between the result of `_get_test_source` and an empty string, which should be true if the function behaves correctly.
- The `Config` class's `repo_root` attribute is not being used or is being modified in some way that affects the test 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/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

Scenario: The test verifies that the `get_test_source_extraction_stops_at_next_def` method stops extracting sources when it reaches a function definition.

Why Needed: This test prevents a potential bug where the source extraction continues beyond the next function definition, potentially causing issues with code readability and maintainability.

Key Assertions:

- The `get_test_source_extraction_stops_at_next_def` method should stop extracting sources when it reaches the next function definition.
- The extracted source code should not be longer than the current line number.
- The test source extraction stops at the end of the first function definition if there are multiple definitions in a file.
- If a function definition is inside another function, the test source extraction should stop before that inner function.
- If there are multiple `with` statements or other block-level constructs inside a function definition, the test source extraction should stop at the end of the last block.
- The extracted source code should not contain any comments or docstrings that would prevent it from being considered part of the source code.
- The test source extraction stops before any nested function definitions or lambda functions.
- If there are multiple `if` statements inside a function definition, the test source extraction should stop at the end of the last statement.

COVERAGE

src/pytest_llm_report/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: Verify that the test source file does not exist when provided.

Why Needed: Prevents a potential bug where the test source file is expected to exist but does not, causing an error or unexpected behavior.

Key Assertions:

- The function _get_test_source returns an empty string when the provided file does not exist.
- The function _get_test_source raises an exception with a meaningful error message when the provided file does not exist.
- The test source file is checked for existence before attempting to retrieve it from the assembler.
- An error message indicating that the file was not found is displayed when the test source file is expected but does not exist.
- A specific error message (e.g. 'FileNotFoundError') is raised when the provided file does not exist.
- The function _get_test_source returns a default value (e.g. '') when the provided file does not exist.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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: Verifies that the `_get_test_source` method correctly extracts a function from a test file with proper indentation.

Why Needed: Prevents a potential bug where the extracted function is not properly indented, leading to incorrect test source coverage.

Key Assertions:

- The `test_file.write_text()` call writes the entire test file contents to the file system.
- The `test_file` path is created with the correct directory structure (`test_example.py`).
- The `write_text()` method appends text to the end of the file, rather than inserting it at the specified indentation level.
- The extracted function has proper indentation (spaces or tabs) according to the test file's syntax.
- The extracted function is not empty and does not contain any commented-out code.
- The `test_file` path is correctly created with a `.py` extension and contains only Python code.
- The `write_text()` method writes text to the end of the file, rather than inserting it at the specified indentation level.

COVERAGE

src/pytest_llm_report/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

Scenario: The 'test_consecutive_lines' test verifies that consecutive lines in a list of integers are compressed into the format 'start-end'.

Why Needed: This test prevents regression where consecutive lines without a specified range are incorrectly compressed to a single integer.

Key Assertions:

- assert compress_ranges([1, 2]) == '1-2'
- assert compress_ranges([1, 3]) == '1-3'
- assert compress_ranges([1, 2, 4]) == '1-4'
- assert compress_ranges([]) == ''
- assert compress_ranges([1]) == '1'
- assert compress_ranges([1, 2, 3, 4]) == '1-4'

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/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 function should correctly identify and return a single range for duplicate values.

Why Needed: This test prevents the function from incorrectly returning multiple ranges for duplicate values in the input list.

Key Assertions:

- assert compress_ranges([1, 2, 2, 3, 3, 3]) == '1-3'
- assert compress_ranges([1, 1, 2, 2, 3, 3]) == '1-3'
- assert compress_ranges([1, 2, 2, 2, 3, 3]) == '1-3'
- assert compress_ranges([1, 1, 1, 2, 2, 3, 3]) == '1-3'
- assert compress_ranges([1, 2, 2, 2, 2, 3, 3]) == '1-3'
- assert compress_ranges([1, 2, 2, 3, 3, 3, 3]) == '1-3'

COVERAGE

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

Why Needed: This test prevents a potential bug where an empty list is not correctly compressed.

Key Assertions:

- The `compress_ranges` function should return an empty string for an empty input list.
- The `compress_ranges` function should handle the case of an empty list by returning an empty string or raising an exception.
- The test should verify that the returned value is indeed 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/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 the function to handle mixed ranges and singles.

Why Needed: This test prevents bugs that may cause incorrect range compression results when dealing with a mix of single values and multiple ranges.

Key Assertions:

- The output should be '1-3, 5, 10-12, 15'.
- The output should contain all the individual numbers from the input list.
- The output should not include any duplicate ranges.
- Any single value in the input list should be included in its own range if it is part of a valid range (e.g., [1, 3] or [10, 15]).
- Any range that spans multiple values should only contain those values and no additional ones.
- The function should handle ranges with different start and end points correctly (e.g., [5, 10] or [12, 15]).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 compressed into a single range.

Why Needed: This test prevents regression where consecutive lines in the input list are not separated by commas.

Key Assertions:

- The function should correctly compress all non-consecutive lines into a single comma-separated string.
- The function should handle cases with one or more consecutive lines that do not need to be compressed.
- The function should keep track of the start and end of each range correctly.
- The function should ignore empty ranges (i.e., ranges with no values).
- The function should handle duplicate values within a single range correctly.
- The function should correctly handle cases where there are multiple consecutive lines that need to be compressed together.
- The function should not compress adjacent lines in the input list if they do not need to be combined.
- The function should preserve the original order of non-consecutive lines when they are compressed.

COVERAGE

src/pytest_llm_report/collector.py	14 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 'single_line' scenario verifies that a single-line input does not utilize the range notation.

Why Needed: This test prevents regression where the function incorrectly handles single-line inputs.

Key Assertions:

- The input list should be returned as a string without any additional characters.
- No error message or exception should be raised for invalid inputs.
- The correct value should be '5' in this case.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: The 'two_consecutive' test verifies that two consecutive lines are compressed to a single range.

Why Needed: This test prevents regression in case of multiple consecutive lines with the same values.

Key Assertions:

- assert compress_ranges([1, 2]) == '1-2'
- assert compress_ranges([3, 4]) == '1-2' or '2-3'
- assert compress_ranges([5, 6]) == '1-2' or '2-3' or '3-4'
- assert compress_ranges([]) == ''
- assert compress_ranges([1]) == '1'
- assert compress_ranges([1, 1]) == '1'
- assert compress_ranges([1, 2, 2]) == '1-2'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 correctly handles an unsorted input range.

Why Needed: This test prevents a potential bug where the function would incorrectly handle unsorted ranges.

Key Assertions:

- Input ranges are sorted in ascending order before compression.
- Compressed ranges are returned as strings in the format 'start-end', e.g. '1-3'.
- The function correctly handles duplicate start values by returning all compressed ranges with that start value.
- Duplicate end values are ignored during range sorting and compression.
- The function preserves the original order of equal start or end values.
- Input ranges with a single element are handled correctly without any issues.
- No unexpected side effects occur when compressing an unsorted input 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: Testing the `expand_ranges` function with an empty string.

Why Needed: This test prevents a potential bug where an empty string would be incorrectly expanded to a non-empty list.

Key Assertions:

- The `expand_ranges` function should return an empty list when given an empty string as input.
- The expected output for the empty string is indeed an empty list.
- Any additional characters in the input string are ignored during expansion.
- No errors or exceptions are raised if the input string is empty.
- The `expand_ranges` function handles edge cases correctly by not attempting to expand a non-string input.
- The test case covers the scenario where the input string is empty, ensuring correctness and robustness.

COVERAGE

src/pytest_llm_report/collector.py	14 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: The test verifies that the `expand_ranges` function correctly handles mixed ranges and singles.

Why Needed: This test prevents a potential bug where the function incorrectly expands single numbers into multiple ranges.

Key Assertions:

- The input string should be parsed into separate ranges or singles.
- A range with a negative start value should be expanded to include all numbers up to that point.
- A range with a positive start value should be expanded from the first number in the range to infinity.
- Single numbers should not be expanded into multiple ranges.
- The function should handle ranges of different lengths correctly.
- The function should return an empty list for invalid input (e.g. no ranges or invalid ranges).
- The function should raise a `ValueError` if it encounters an invalid 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	11 lines (ranges: 81, 84-91, 93, 95)

PASSED tests/test_ranges.py::TestExpandRanges::test_range 1ms 3

AI ASSESSMENT

Scenario: The `expand_ranges` function is expected to correctly handle a range input.

Why Needed: This test prevents the function from expanding the range incorrectly when negative numbers are provided.

Key Assertions:

- The function should return a list of integers between 1 and the maximum number in the range.
- The function should not expand the range beyond the specified end value.
- The function should handle ranges with multiple elements correctly.
- The function should ignore non-integer values within the range.
- The function should raise an error for invalid input (e.g. non-string or non-integers).
- The function should preserve the original order of numbers in the range.
- The function should support negative ranges (e.g. -3-5).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Test 'compress_ranges and expand_ranges should be inverses' verifies that the inverse relationship between 'compress_ranges' and 'expand_ranges' is maintained.

Why Needed: This test prevents regression in the range compression and expansion functionality, ensuring that compressing a list of numbers returns the original list when expanded back to its original form.

Key Assertions:

- The function `expand_ranges(compressed)` should return `original` when called with `compressed` as an argument.
- The function `compress_ranges(original)` should be able to correctly decompress and expand a compressed range.
- When the input list is empty, both `expand_ranges(compressed)` and `compress_ranges(original)` should return `[]`.
- When the input list contains only one element, `expand_ranges(compressed)` should return `[1, 2, 3]` and `compress_ranges(original)` should return `[1, 2, 3]`.
- For a range with an odd number of elements, `expand_ranges(compressed)` should return the original list when expanded back to its original form.
- The function `expand_ranges([1, 2, 3])` should be able to correctly expand a single-element range.
- The function `compress_ranges([1, 2, 3, 4, 5])` should be able to compress a list with an odd number of elements and then decompress it back to its original form.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 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 (a single number) and return a list with that number as the only element.

Why Needed: This test prevents regression where the function does not produce the correct result for a single input, potentially causing unexpected behavior or errors in downstream code.

Key Assertions:

- The 'expand_ranges' function should return a list containing the input number.
- The 'expand_ranges' function should only contain one element in the returned list.
- The 'expand_ranges' function should not raise any exceptions for single inputs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Tests the test_milliseconds function to verify it formats durations as milliseconds for < 1s.

Why Needed: This test prevents a potential regression where the function does not correctly format durations in milliseconds for values less than 1 second.

Key Assertions:

- The function should return '500ms' when given duration 0.5 seconds.
- The function should return '1ms' when given duration 0.001 seconds.
- The function should return '0ms' when given duration 0.0 seconds.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 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 the format_duration function with seconds input.

Why Needed: Prevents a potential bug where the function does not correctly format durations for seconds or minutes.

Key Assertions:

- The function should return '1.23s' when given an input of 1.23 seconds.
- The function should return '60.00s' when given an input 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: This test prevents regression in the outcome_to_css_class function when passing different types of outcomes (e.g., 'xfailed' instead of just 'failed')

Key Assertions:

- outcome-to-css-class mapping is correct for all possible outcomes ('passed', 'failed', 'skipped', 'error')
- outcome_to_css_class('passed') returns the expected string ('outcome-passed')
- outcome_to_css_class('failed') returns the expected string ('outcome-failed')
- outcome_to_css_class('skipped') returns the expected string ('outcome-skipped')
- outcome_to_css_class('xfailed') returns the expected string ('outcome-xfailed')
- outcome_to_css_class('xpassed') returns the expected string ('outcome-xpassed')
- outcome_to_css_class('error') returns the expected string ('outcome-error')

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 regression where the 'outcome' parameter is not recognized by the `outcome_to_css_class` function.

Key Assertions:

- The function should return 'outcome-unknown' when the 'outcome' parameter is set to 'unknown'.
- The function should raise an error if the 'outcome' parameter is not recognized ('unknown').
- The function should return 'outcome-unknown' for any other unrecognized outcome.
- The function should handle cases where the 'outcome' parameter is a string that cannot be converted to a number or boolean.
- The function should throw an exception when the 'outcome' parameter is set to a non-string value.
- The function should return 'outcome-unknown' for any other unrecognized outcome.
- The function should handle cases where the 'outcome' parameter is set to 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/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: The test verifies that a basic report is rendered correctly with the expected HTML structure.

Why Needed: This test prevents a rendering issue where the report does not display as expected, potentially leading to incorrect interpretation of test results.

Key Assertions:

- The presence of a complete HTML document.
- The inclusion of "Test Report" in the HTML.
- The identification of specific nodes with outcomes and error messages.
- The verification of plugin and repository information.
- The display of all test cases, including passed and failed ones.

COVERAGE

src/pytest_llm_report/collector.py	14 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: Verify that the test renders coverage information and includes a specific file path.

Why Needed: This test prevents a regression where coverage information is not included or reported correctly.

Key Assertions:

- The report root contains a 'tests' list with a single 'TestCaseResult' node.
- The 'coverage' field in the 'testCase_result' node contains a 'file_path' attribute matching 'src/foo.py'.
- The 'line_ranges' and 'line_count' attributes of the 'coverage_entry' node match the expected values (1-5 lines, 5 lines).

COVERAGE

src/pytest_llm_report/collector.py	14 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: Test renders LLM annotation for fallback HTML scenario.

Why Needed: This test prevents a security vulnerability where an attacker could manipulate the LLM annotations to bypass authentication.

Key Assertions:

- The report includes the 'Tests login flow' scenario in its HTML.
- The report includes the 'Prevents auth bypass' why needed message in its HTML.
- The 'Tests login flow' and 'Prevents auth bypass' assertions are present in the rendered HTML.

COVERAGE

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

Scenario: Verifies that the source coverage summary is included in the rendered HTML.

Why Needed: This test prevents a regression where the source coverage summary was not displayed correctly due to missing or incomplete coverage.

Key Assertions:

- The 'Source Coverage' section should be present in the rendered HTML.
- The 'src/foo.py' file path should be included in the 'Source Coverage' section.
- The percentage of covered code should be displayed as '80.0%' in the 'Source Coverage' section.
- All statements in the source code should be covered by at least 50% to meet the coverage threshold.
- Only missed statements should be reported as 'missed' in the source coverage summary.
- The missing ranges should be correctly formatted as '1-4, 6-8' in the source coverage summary.
- The missed ranges should not exceed the total number of lines in the file (10) in the source coverage summary.
- All covered ranges should be correctly formatted as '5, 9-10' in the source coverage summary.
- Only statements that are actually covered should be reported as 'covered' in the source coverage summary.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/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 'XFailed' and 'XPassed' summary entries are included in the rendered HTML report.

Why Needed: This test prevents a regression where the summary section is missing or incorrectly formatted when using XPass.

Key Assertions:

- The string 'XFailed' should be present in the rendered HTML report.
- The string 'XPassed' should be present in the rendered HTML report.
- Both 'XFailed' and 'XPassed' summary entries should be included in the rendered HTML report.

COVERAGE

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

Scenario: Test verifies that different content produces different hashes.

Why Needed: This test prevents a potential bug where the same input could produce the same hash value, potentially leading to incorrect reporting or analysis of data.

Key Assertions:

- The output of `compute_sha256(b'hello')` should be different from `compute_sha256(b'world')`.
- The output of `compute_sha256(b'hello')` should not match the expected hash value.
- The output of `compute_sha256(b'world')` should not match the expected hash value.
- The output of `compute_sha256(b'hello')` should be a different SHA-256 hash than `compute_sha256(b'world')`.
- The output of `compute_sha256(b'hello')` should have a different hexadecimal representation than `compute_sha256(b'world')`.
- The output of `compute_sha256(b'hello')` should not be equal to the expected hash value for `compute_sha256(b'world')`.
- The output of `compute_sha256(b'hello')` should have a different hexadecimal representation than the expected hash value for `compute_sha256(b'world')`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Testing the empty bytes case for consistency and correctness of the SHA-256 hash.

Why Needed: This test prevents a potential bug where an empty byte string could produce different hashes, potentially leading to incorrect reporting or analysis.

Key Assertions:

- The two computed hashes should be equal (i.e., they should have the same value).
- Both computed hashes should have a length of 64 bytes (the expected SHA-256 hex length).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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 6ms 4

AI ASSESSMENT

Scenario: Test that the build_run_meta method returns the expected metadata for a test run.

Why Needed: This test prevents regression where the report writer does not include version info in the meta data of a test run.

Key Assertions:

- The duration of the test run 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__.
- The ReportWriter instance should have a valid _build_run_meta method call with expected 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	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 verifies that the `build_summary` method counts all outcome types correctly.

Why Needed: This test prevents a regression where the summary does not accurately count all outcome types.

Key Assertions:

- The total number of outcomes should be 6 (passed, failed, skipped, xfailed, xpassed, error).
- The passed outcome should have a value of 1.
- The failed outcome should have a value of 1.
- The skipped outcome should have a value of 1.
- The xfailed and xpassed outcomes should each have a value of 1.
- The error outcome should have a value of 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 `total` attribute of the `Summary` object is correctly calculated by summing up all outcomes.

Why Needed: This test prevents a regression where the total count of outcomes in the summary might be incorrect due to missing or incorrectly counted failed tests.

Key Assertions:

- The total count should match the number of passed tests (2).
- The total count should match the number of failed tests (1).
- The total count should match the number of skipped tests (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`	`13 lines (ranges: 156-158, 312, 314-315, 317-322, 330)`

PASSED tests/test_report_writer.py::TestReportWriter::test_create_writer 1ms 4

AI ASSESSMENT

Scenario: Test creates a new ReportWriter instance and verifies its initial state.

Why Needed: This test prevents a potential bug where the Writer's configuration is not properly initialized.

Key Assertions:

- The `config` attribute of the `ReportWriter` instance should be set to the provided `Config` object.
- The `warnings` list of the `ReportWriter` instance should be empty.
- The `artifacts` dictionary of the `ReportWriter` instance should be 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/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` 6ms 4

AI ASSESSMENT

Scenario: Test 'ReportWriter::test_write_report_assembles_tests' verifies that the report includes all tests.

Why Needed: This test prevents a regression where the report does not include all tests, potentially leading to incomplete or misleading information.

Key Assertions:

- The length of the report.tests list should be equal to 2.
- The total count of tests in the summary should be equal to 2.
- Each test result should have a 'nodeid' attribute matching 'test1'.
- Each test result should have an 'outcome' attribute matching either 'passed' or 'failed'.
- The report.summary.total property should have a value equal to 2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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` 7ms 4

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class writes a report with an included coverage percentage.

Why Needed: This test prevents a bug where the coverage percentage is not included in the report, potentially leading to incorrect reporting.

Key Assertions:

- The total coverage percentage of all files covered by the test should be equal to or greater than the specified `coverage_percent` value.
- The `summary.coverage_total_percent` attribute should contain the specified `coverage_percent` value.
- All files included in the report should have a total coverage percentage that is equal to or greater than the specified `coverage_percent` value.
- No file should have a total coverage percentage less than the specified `coverage_percent` value.
- The coverage percentage of each individual file should be calculated correctly and added to the total.
- All files included in the report should have their coverage percentages summed up 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	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` 6ms 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/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` 6ms 4

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_merges_coverage verifies that the report writer merges coverage into tests.

Why Needed: This test prevents regression where the report writer does not merge coverage correctly, potentially leading to incorrect reporting of test coverage.

Key Assertions:

- The number of tests in the report should be 1.
- Each test's file path should match the expected file path (src/foo.py).
- Each test's coverage data should have a single entry with the correct file path and line ranges.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 8ms 5

AI ASSESSMENT

Scenario: Test the ReportWriter with a direct write failure scenario.

Why Needed: This test prevents a regression where the atomic write fails and the fallback to direct write is not applied.

Key Assertions:

- The report.json file should exist at the specified path.
- Any warnings in the writer should have a code of 'W203'.
- The writer's warnings list should contain any warning with code 'W203'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	67 lines (ranges: 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` 8ms 5

AI ASSESSMENT

Scenario: The 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 fails to create the output directory if it already exists.

Key Assertions:

- The output directory is created at the specified path.
- The file `report.json` is written to the output directory.
- The existence of the output directory is checked before writing the report.
- An error is raised when attempting to write the report in an existing directory.
- The test fails if the output directory already 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/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` 2ms 4

AI ASSESSMENT

Scenario: Test ensures that directory creation fails and raises a warning.

Why Needed: This test prevents a potential bug where the report writer does not raise an error when creating a non-existent directory.

Key Assertions:

- The 'mkdir' function is called with a side effect of raising an OSError exception.
- Any warnings raised by the ReportWriter are checked to have a code of 'W201'.
- The 'writer.warnings' list contains at least one warning object with a code of 'W201'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 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 that the `get_git_info` function handles git command failures gracefully by returning `None` for both SHA and dirty flag.

Why Needed: This test prevents a regression where the `get_git_info` function fails to return expected values when encountering a git command failure.

Key Assertions:

- The `get_git_info` function should return `None` for both `sha` and `dirty` variables.
- The `get_git_info` function should not raise an exception when encountering a git command failure.
- The `get_git_info` function should handle the case where `git` is not found without raising an exception.
- The `get_git_info` function should return `None` for both `sha` and `dirty` variables even if `git` is found but returns a non-zero exit status.
- The `get_git_info` function should not raise an exception when encountering a git command failure, instead returning `None` as expected.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 38ms 6

AI ASSESSMENT

Scenario: Test 'test_write_html_creates_file' verifies that the report writer creates an HTML file.

Why Needed: This test prevents a regression where the report writer fails to create an HTML file even when there are tests with failed or skipped nodes.

Key Assertions:

- The file should exist at the specified path.
- The report should contain expected content as described in the assertions.
- All relevant node types (test1, test2) should be present in the report.
- The report should include the correct status indicators (PASSED, FAILED, SKIPPED, XFailed, XPassed, Errors).
- The HTML file should contain all necessary information to recreate the test environment.

COVERAGE

src/pytest_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` 38ms 6

AI ASSESSMENT

Scenario: Test `test_write_html_includes_xfail_summary` verifies that the report includes xfail outcomes in the HTML summary.

Why Needed: This test prevents a regression where xfail outcomes are not included in the HTML summary, which can make it difficult to identify failed tests.

Key Assertions:

- The string 'XFAILED' is present in the HTML output.
- The string 'XFailed' is present in the HTML output.
- The string 'XPASSED' is present in the HTML output.
- The string 'XPassed' is present in the HTML output.
- The report includes all xfail outcomes (nodeid='test_xfail', outcome='xfailed') and all pass outcomes (nodeid='test_xpass', outcome='xpassed').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 7ms 5

AI ASSESSMENT

Scenario: Test verifies that a JSON file is created with the report.

Why Needed: This test prevents regression where the `ReportWriter` does not create a JSON file.

Key Assertions:

- The file should exist at the specified path.
- The `report.json` artifact should be tracked by the writer.

COVERAGE

src/pytest_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` 43ms 6

AI ASSESSMENT

Scenario: Test that `write_pdf` creates a PDF file when Playwright is available.

Why Needed: Prevents regression where the test fails to create a PDF file even if Playwright is installed.

Key Assertions:

- The `write_pdf` function of the `ReportWriter` class should write the report to a file at the specified path.
- The `report.pdf` attribute of the `writer` object should contain the path to the created PDF file.
- Any artifacts generated by the test should have a path that matches the path of the created PDF file.
- The `exists()` method of the `pdf_path` variable should return True.
- The `any()` function with a generator expression should find any artifact with a path matching the PDF file path.
- The `writer.artifacts` list should contain an artifact with a path equal to the PDF file path.
- The `writer.artifacts` list should not be empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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` 7ms 4

AI ASSESSMENT

Scenario: Test that a warning is raised when the Playwright library is missing for PDF output.

Why Needed: To prevent a potential issue where a PDF report is generated without the required Playwright library.

Key Assertions:

- The file 'report.pdf' should not exist.
- Any warnings with code WarningCode.W204_PDF_PLAYWRIGHT_MISSING should be present.

COVERAGE

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

AI ASSESSMENT

Scenario: Verifies that `get_git_info` returns `None` when the input path does not exist.

Why Needed: Prevents a potential bug where `get_git_info` incorrectly reports Git info for non-existent paths.

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 report any Git info (e.g., commit hash, file contents) for a nonexistent path.
- The test assertion `assert dirty is None` should be removed or updated to reflect the correct behavior.
- The test assertion `assert sha is None` should also be removed or updated to reflect the correct behavior.
- The function `get_git_info(tmp_path)` should raise an exception when called with a nonexistent path, rather than returning `None` and silently failing the test.
- The test should include additional assertions to verify that the function behaves correctly for non-existent paths, such as checking if the function raises an exception or returns an error message.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/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` 6ms 3

AI ASSESSMENT

Scenario: Test getting git info from a valid git repo.

Why Needed: Prevents test failure due to unexpected behavior when running tests in a non-git repository.

Key Assertions:

- The function `get_git_info` should return either the SHA of the Git repository or an empty string if it's not found.
- The function `get_git_info` should not crash unexpectedly if the test is run in a non-git repository.
- The function `get_git_info` should correctly handle cases where the repository is not found (i.e., return None).
- The function `get_git_info` should raise an exception with a meaningful error message when the repository is not found.
- The function `get_git_info` should log a warning or informational message indicating that the test was run in a non-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 correctly when the import fails.

Why Needed: This test prevents a regression where plugins' Git info would not be available due to an import failure.

Key Assertions:

- The function `get_plugin_git_info()` should return `None` or a string if `_git_info` is missing from the cache.
- The function `get_plugin_git_info()` should still work via the git runtime fallback even when `_git_info` is missing from the cache.
- The function `get_plugin_git_info()` should not raise an exception when `_git_info` is missing from the cache.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: This test prevents a potential bug where the function `get_plugin_git_info()` crashes due to an invalid input.

Key Assertions:

- The function `get_plugin_git_info()` should return either None or a string.
- The function `get_plugin_git_info()` should not crash if it receives an invalid input.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 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: Prevents regression in case of unexpected report generation errors.

Key Assertions:

- The test file exists at the expected location after writing a report.
- The report is not empty.
- No exceptions are raised during the write process.
- The writer does not attempt to use atomic writes for this test scenario.
- The direct write path is used instead of an atomic write.
- The report generation process completes successfully 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	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` 121ms 6

AI ASSESSMENT

Scenario: Test PDF generation with playwright exception (lines 424-432)

Why Needed: Prevents regression when playwright raises an exception during PDF generation.

Key Assertions:

- Mocking playwright as installed but failing should raise a RuntimeError.
- The `write_pdf` method should return a warning indicating PDF failure.
- The `warnings` attribute of the writer instance should contain any 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/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 that a report is generated without using the playwright when it's not installed.

Why Needed: This test prevents a potential bug where reports are generated incorrectly or fail to create due to missing playwright dependencies.

Key Assertions:

- The 'W204' warning should be present in the report's warnings list.
- The report file should not exist.
- The report should have been created by ReportWriter without using playwright.

COVERAGE

src/pytest_llm_report/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` 31ms 6

AI ASSESSMENT

Scenario: Test _resolve_pdf_html_source creates temp file when no HTML source is provided.

Why Needed: Prevents a potential issue where the test fails to create a temporary PDF file if no HTML source is provided.

Key Assertions:

- The path created by _resolve_pdf_html_source should exist and have a suffix of '.html'.
- The path created by _resolve_pdf_html_source should not be empty.
- The path created by _resolve_pdf_html_source should have the correct file extension (.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/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` 32ms 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 incorrect assumption about the existence of the HTML file.

Key Assertions:

- The path returned by _resolve_pdf_html_source should be a temporary file.
- The path returned by _resolve_pdf_html_source should exist after calling resolve_pdf_html_source.
- The report writer will fall back to using a temporary file if the configured HTML does not exist.
- The report writer will create a new temporary file instead of deleting the existing one.
- The report writer will preserve the original configuration settings when resolving the PDF.
- The report writer will handle the case where the configured HTML is an empty string or None.
- The report writer will use the configured path for the HTML source even if it's not a valid file.
- The test will fail with an assertion error if the reported path does not match the expected one.

COVERAGE

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

Key Assertions:

- The path of the resolved HTML file matches the provided HTML file.
- The `is_temp` flag is set to False, indicating that the report was generated successfully without leaving a temporary file behind.

COVERAGE

src/pytest_llm_report/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` 2ms 4

AI ASSESSMENT

Scenario: Verify that directory creation is prevented when the report HTML file already exists.

Why Needed: To prevent a potential directory traversal attack by ensuring the report HTML file does not get deleted before the test finishes.

Key Assertions:

- The `tmp_dir / 'r.html'` path should exist after running the test.
- Any warnings with code 'W202' should be present in the report HTML file.
- The directory created by the report writer should not have been deleted during 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/report_writer.py	11 lines (ranges: 156-158, 470-477)

PASSED `tests/test_report_writer_coverage_v2.py::test_report_writer_metadata_skips` 12ms 5

AI ASSESSMENT

Scenario: The test verifies that the report writer skips metadata when reports are disabled.

Why Needed: This test prevents a bug where metadata is not included in the report even when reports are disabled.

Key Assertions:

- Metadata should be present in the run meta.
- LLM model should be absent from the run meta.
- Start time should be present in the run meta.
- LLM model should not be present in the run meta (when report is disabled).
- Start time and LLM model should both be present in the run meta (when report 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/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 that `AnnotationSchema.from_dict` can create a valid annotation from a dictionary.

Why Needed: Prevents regression in handling malformed input data.

Key Assertions:

- assert schema.scenario == 'Verify login'
- assert schema.why_needed == 'Catch auth bugs'
- assert schema.key_assertions == ['assert 200', 'assert token']
- assert schema.confidence == 0.95

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

PASSED tests/test_schemas.py::TestAnnotationSchema::test_to_dict_full 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `to_dict` method of the `AnnotationSchema` class correctly converts a schema to a dictionary, including all required fields.

Why Needed: This test prevents regression in the `AnnotationSchema` class where it fails to convert a schema with all required fields to a dictionary.

Key Assertions:

- assert data['scenario'] == 'Verify login',
- assert data['why_needed'] == 'Catch auth bugs',
- assert data['key_assertions'] == ['assert 200', 'assert token'],
- assert data['confidence'] == 0.95

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	8 lines (ranges: 90-92, 94-98)
src/pytest_llm_report/plugin.py	6 lines (ranges: 363-364, 367, 371-373)

📄 tests/test_smoke_pytester.py

15 tests

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_report_created` 100ms 8

AI ASSESSMENT

Scenario: Test that an HTML report is created when the test function `test_simple` is run.

Why Needed: This test prevents a regression where the HTML report is not generated for tests with simple functions.

Key Assertions:

- The file `report.html` should exist in the current working directory.
- The string `` should be present in the content of the `report.html` file.
- The string `test_simple` should be present in the content of the `report.html` 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	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` 137ms 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 a report may not display all statuses due to missing or incorrect summary labels.

Key Assertions:

- The function `assert_summary(labels: list[str], expected: int)` checks if the count of each status matches the expected value in the HTML report.
- It also checks for missing summary labels by searching for any non-numeric text that could be a label.
- If it finds a match, it raises an AssertionError with a message indicating which label is missing and why it's missing.
- The function `assert_summary(labels: list[str], expected: int)` also checks if the count of each status matches the expected value in the HTML report.
- It also checks for missing summary labels by searching for any non-numeric text that could be a label.
- If it finds a match, it raises an AssertionError with a message indicating which label is missing and why it's missing.
- The function `assert_summary(labels: list[str], expected: int)` also checks if the count of each status matches the expected value in the HTML report.
- It also checks for missing summary labels by searching for any non-numeric text that could be a label.
- If it finds a match, it raises an AssertionError with a message indicating which label is missing and why it's missing.

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` 76ms 7

AI ASSESSMENT

Scenario: The JSON report is created successfully.

Why Needed: This test prevents a bug where the report is not generated correctly due to an incorrect path or file name.

Key Assertions:

- report_path.exists()
- data['schema_version']
- data['summary']['total'] == 2
- data['summary']['passed'] == 1
- data['summary']['failed'] == 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` 68ms 13

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report when a provider is enabled.

Why Needed: Prevent regressions by ensuring LLM annotations are present in the report.

Key Assertions:

- The 'why_needed' assertion checks if the LLM annotation is necessary to prevent regressions.
- The 'key_assertions' list includes critical checks performed during the test, such as asserting True.
- The 'pytester.makepyfile()' function creates a test case with an enabled provider and mock completion.
- The 'pytester.makeconftest()' function patches litellm.completion before it's imported.
- The 'pytester.makefile()' function creates a pyproject.toml file with the [tool.pytest_llm_report] configuration.
- The 'pytester.makepyfile()' function includes an enabled provider in the test case.
- The 'pytester.makeconftest()' function patches litellm.completion before it's imported, ensuring LLM annotations are included.

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` 99ms 13

AI ASSESSMENT

Scenario: Verify that LLM errors are reported in the HTML output.

Why Needed: Prevent regression where LLM errors are not surfaced in the report.

Key Assertions:

- The test verifies that the `pytest_llm_error_is_reported` function is called with an error message.
- The test ensures that the error message is displayed correctly in the HTML output.
- The test checks that the error message is reported for LLM errors.
- The test verifies that the error message is not suppressed or hidden from view.
- The test checks that the error message is displayed with a clear and descriptive title.
- The test ensures that the error message is correctly formatted and styled in the report.
- The test verifies that the error message is reported after all tests have completed.

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 66ms 7

AI ASSESSMENT

Scenario: Test the LLM opt-out marker.

Why Needed: Prevents regression in test cases where LLM opt-out is expected to be recorded.

Key Assertions:

- The 'llm_opt_out' marker should be present in the test file.
- The 'llm_opt_out' marker should be set to True for all tests in the report.
- The number of tests in the report should be exactly one.
- All tests in the report should have an 'llm_opt_out' key with a value of True.
- The 'llm_opt_out' key should not be missing from any test data.
- The 'llm_opt_out' marker should be set to True for all tests in the report, even if they are empty.
- The 'llm_opt_out' marker should be set to True for all tests in the report, even if there are no assertions.

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 63ms 7

AI ASSESSMENT

Scenario: Test the requirement marker to ensure it is correctly recorded and reported.

Why Needed: This test prevents a regression where the requirement marker might not be properly recorded or reported, potentially leading to incorrect test results.

Key Assertions:

- The `@pytest.mark.requirement` decorator should record the specified requirements.
- The report generated by `--llm-report-json` should include the required requirements.
- The report should contain only one requirement.
- The first requirement in the report should be 'REQ-001'.
- The second requirement in the report should be 'REQ-002'.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-200, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	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` 75ms 7

AI ASSESSMENT

Scenario: Test 'Multiple xfailed tests are recorded in the report' verifies that multiple xfailed tests are correctly reported.

Why Needed: This test prevents regression by ensuring that when a test is marked as xfailed, it will be included in the report and counted towards the total number of failed tests.

Key Assertions:

- The 'summary' key in the JSON report contains the correct value for 'xfailed', which should be 2.
- Each test in the 'tests' list has an 'outcome' attribute that is either 'xfailed' or 'not xfailed'.
- When a test is marked as xfailed, its outcome is included in the 'outcomes' list and matches the expected value of ['xfailed', 'xfailed'].
- The total number of failed tests reported in the JSON report should match the actual count from running pytest.
- No other tests are recorded in the report when a test is marked as xfailed, preventing false positives.
- When multiple tests are marked as xfailed and run concurrently, they will be correctly counted towards the total number of failed tests.
- The 'summary' key in the JSON report contains the correct value for 'xfailed', which should be 2.

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 64ms 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 propagated to the report.

Key Assertions:

- The number of skipped tests should be exactly 1.
- The 'summary' key in the report data should contain a single value: 'skipped'.
- The 'skipped' value in the report data should be an integer (not a string).

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 68ms 7

AI ASSESSMENT

Scenario: Test 'Xfailed tests are recorded' verifies that xfailed tests are correctly counted in the report.

Why Needed: This test prevents a regression where xfailed tests are not reported correctly due to an issue with the pytester.runpytest function.

Key Assertions:

- The function `pytester.makepyfile()` is called with the correct arguments.
- The `--llm-report-json` argument is set correctly on the `pytester.path /
- The `pytester.runpytest()` function runs without any errors or exceptions.
- The report generated by `pytester.runpytest()` contains the correct data.
- The 'xfailed' key in the report data dictionary has a value of 1.
- The number of xfailed tests is correctly reported in the report.

COVERAGE

src/pytest_llm_report/collector.py	47 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-116, 119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212, 216, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	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` 69ms 7

AI ASSESSMENT

Scenario: Test 'test_parametrized_tests' verifies that parameterized tests are recorded separately.

Why Needed: This test prevents regression by ensuring that the correct number of parameterized tests are run and reported.

Key Assertions:

- The total count of parameterized tests is 3 (1 passed, 2 failed).
- Each parameterized test has a unique input value 'x'.
- The report path contains the expected data structure with 'summary' and 'passed' keys.
- The number of successful parameterized tests is greater than or equal to the number of failed ones.

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` 55ms 3

AI ASSESSMENT

Scenario: The CLI help text should include usage examples.

Why Needed: This test prevents a potential bug where the user is not provided with clear instructions on how to use the LLM report feature.

Key Assertions:

- The output of `pytester.runpytest('--help')` contains at least one line that matches the pattern `*Example:*--llm-report*"
- The output does not contain any lines that suggest how to use the LLM report feature
- The test fails if the user is not provided with clear instructions on how to use the LLM report feature

COVERAGE

src/pytest_llm_report/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` 54ms 3

AI ASSESSMENT

Scenario: Verify that LLM markers are registered and their presence is detected by pytester.

Why Needed: This test prevents a potential bug where the 'markers' marker is not recognized or registered by pytest, potentially leading to unexpected behavior or errors in subsequent tests.

Key Assertions:

- The 'markers' marker should be present in the stdout of the runpytest command.
- The 'llm_opt_out*' line should match the expected output.
- The 'llm_context*' line should also match the expected output.
- The 'requirement*' line should match the expected output.
- The 'markers' marker should be recognized by pytest and registered 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	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` 54ms 3

AI ASSESSMENT

Scenario: Test that the plugin is registered correctly by running `pytest --help` and verifying the output.

Why Needed: This test prevents a potential bug where the plugin registration fails or is not detected by pytest11, leading to incorrect reporting.

Key Assertions:

- The output of `pytest --help` should contain the string 'llm-report' in the first line.
- The output of `pytest --help` should be a valid Python script with no syntax errors.
- The output of `pytest --help` should not contain any plugin-related warnings or errors.
- The output of `pytest --help` should display the expected help message for the 'llm-report' command.
- The output of `pytest --help` should be a valid Python script with no syntax errors and proper indentation.
- The output of `pytest --help` should not contain any plugin-related warnings or errors, even if there are plugins installed.
- The output of `pytest --help` should display the expected help message for the 'llm-report' command without any additional information.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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` 96ms 8

AI ASSESSMENT

Scenario: Test that special characters in nodeid do not cause a crash or invalid HTML report.

Why Needed: This test prevents potential crashes and ensures the validity of the generated HTML report.

Key Assertions:

- The '<' character is present in the provided string.
- The '>' character is present in the provided string.
- The '&' character is present in the provided string.
- The 'html' tag is present in the generated HTML content.
- The '' tag is present in the generated HTML content.

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 function `format_duration` with a boundary of exactly one minute.

Why Needed: This test prevents regression where the function might incorrectly format durations less than one minute.

Key Assertions:

- The result should be '1m 0.0s' for a duration of 60 seconds.
- The result should not include any units other than 'm' and 's'.
- The time part should be exactly one unit (either 'm' or 's').
- The seconds part should be zero.
- The function should handle durations less than one minute correctly.
- The function should return the correct format for a duration of 1 second.
- The function should not silently fail for invalid input (e.g. negative numbers).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Tests the `format_duration` function to ensure it correctly formats sub-millisecond durations as microseconds.

Why Needed: This test prevents a potential bug where the function does not handle microsecond values correctly, potentially leading to incorrect formatting or errors in output.

Key Assertions:

- The result of `format_duration(0.0005)` contains the string 'µs' (microseconds).
- The assertion `result == '500µs'` is used to verify that the formatted string matches the expected value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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: Test formats sub-second durations as milliseconds.

Why Needed: Prevents regression when the test duration is not a whole number of seconds.

Key Assertions:

- assert 'ms' in result
- assert result == '500.0ms'
- result should be exactly equal to 500.0 ms

COVERAGE

src/pytest_llm_report/collector.py	14 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: Test the 'minutes' unit of duration to ensure it is correctly formatted.

Why Needed: This test prevents a potential bug where the format string does not include the 'm' suffix for durations over one minute.

Key Assertions:

- The result should contain the string '1m' to indicate minutes.
- The result should contain the string '30.5s' to represent the seconds part of the duration.
- The format string should be able to correctly handle durations over one minute, including the 'm' suffix.

COVERAGE

src/pytest_llm_report/collector.py	14 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 duration of 3 minutes and 5 seconds.

Why Needed: This test prevents regression in the 'format_duration' function when given durations with multiple minutes.

Key Assertions:

- The result should be '3m 5.0s'.
- The format string '{:.2f}m {}s' is used correctly to display the duration.
- The function does not throw any exceptions for invalid input (e.g., negative numbers).

COVERAGE

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

Why Needed: Prevents a potential bug where the function returns an incorrect time representation for a single-second duration.

Key Assertions:

- The output should be '1.00s' (one second and zero seconds).
- The output should not contain any trailing zeros.
- The function should handle cases where the input is a float greater than or equal to 10.0.
- The function should handle cases where the input is a float less than 10.0.
- The function should return an error for inputs outside the range [1, 100].
- The function should preserve the correct order of magnitude in the output.
- The function should not silently fail when given very large or very small inputs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: Verifies the 'seconds' unit is included in the formatted duration string.

Why Needed: Prevents a potential bug where seconds are not correctly formatted as 'xx.xxxx' strings.

Key Assertions:

- The function `format_duration(5.5)` returns a string that includes the 's' unit.
- The function `format_duration(5.5)` returns the correct formatted string '5.50s'.
- The function `format_duration(5.5)` correctly handles decimal numbers with two digits after the decimal point.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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: Verifies the correct formatting of small millisecond durations.

Why Needed: This test prevents a potential bug where millisecond values are not correctly formatted as '1.0ms'.

Key Assertions:

- The function `format_duration(0.001)` returns the string '1.0ms' for a duration of 1 millisecond.
- The function `format_duration(x)` returns the correct formatting of x milliseconds.
- The test asserts that the result is equal to '1.0ms'.
- The test checks that the assertion passes when x = 0.001.

COVERAGE

src/pytest_llm_report/collector.py	14 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 duration of 1 microsecond.

Why Needed: This test prevents a potential bug where the function incorrectly formats very small durations as seconds instead of microseconds.

Key Assertions:

- The function should return '1µs' when given a duration of 0.000001 seconds.
- The function should not format the duration as seconds but rather microseconds.
- The function should handle negative values correctly and still return the correct result.
- The function should be able to handle very small durations without losing precision.
- The function should support all possible formats for durations (e.g., 1ms, 2s, 3µs).
- The test should fail when given a duration that is not in microseconds or seconds.
- The test should pass with the correct result even if the input value is very close to zero but not exactly zero.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Test the functionality of datetime with UTC timezone in ISO format.

Why Needed: This test prevents a potential bug where the datetime is not correctly formatted with UTC timezone.

Key Assertions:

- The datetime object should be created with the UTC timezone.
- The resulting string should match '2024-01-15T10:30:45+00:00'.
- The time zone offset should be correctly calculated as +00:00.
- The date part of the datetime object should remain unchanged.
- The AM/PM designation should not be affected by the timezone.
- The resulting string should have a length of 19 characters (YYYY-MM-DDTHH:MM:SS+HHMM).
- No exceptions or errors should occur during the test execution.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 bug where the naive datetime format is incorrectly assumed to be in UTC.

Key Assertions:

- The output of `iso_format(dt)` should be '2024-06-20T14:00:00'.
- The datetime object `dt` passed to `iso_format()` has no timezone.
- The `datetime` class is used without specifying a timezone.
- The `iso_format()` function returns the correct output for naive datetime without timezone.
- The `assert` statement checks if the result matches the expected value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	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: Test the functionality of formatting a datetime object with microseconds.

Why Needed: This test prevents a potential issue where microseconds are not correctly formatted as ISO 8601.

Key Assertions:

- The output string should contain the value '123456' which represents the microseconds part of the datetime.
- The microseconds part should be in the format '000-99', i.e., '00:01:30'.
- The microseconds part should not exceed 3 digits.
- The microseconds part should not be negative.
- The microseconds part should only contain digits and can include leading zeros for correct formatting.
- The microseconds part should not have a decimal point or any other non-numeric characters.
- The microseconds part should be correctly formatted as per ISO 8601 standard.

COVERAGE

src/pytest_llm_report/collector.py	14 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

Scenario: Verifies that the `utc_now()` function returns a datetime object with an associated UTC timezone.

Why Needed: Prevents regression in cases where the system does not have a valid UTC timezone.

Key Assertions:

- The returned datetime object has a non-null `tzinfo` attribute and is equal to `UTC`.
- The `tzinfo` attribute of the returned datetime object is set to `UTC`.
- The returned datetime object's timezone is indeed `UTC`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 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: Verifies that the `utc_now()` function returns a current time within a specified tolerance.

Why Needed: This test prevents a potential bug where the returned time is not accurate due to the UTC offset being outdated.

Key Assertions:

- The `before` variable represents the current time before calling `utc_now()`.
- The `result` variable represents the current time after calling `utc_now()`.
- The `after` variable represents the current time after calling `utc_now()`.
- The `before <= result <= after` condition ensures that the returned time is within a specified tolerance (e.g., 1 second).
- The `assert before <= result <= after` statement verifies that the returned time meets this condition.
- If the UTC offset is outdated, the returned time may not be accurate. This test prevents such bugs by ensuring that the returned time is always within the expected 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/time.py	1 lines (ranges: 15)

PASSED `tests/test_time.py::TestUtcNow::test_returns_datetime` 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)

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 when get-token command fails.

Why Needed: This test prevents a bug where the TokenRefresher class does not raise an exception when it encounters a command that fails to authenticate.

Key Assertions:

- The 'exit 1' status code is expected in the pytest.raises() context.
- The string 'Authentication failed' is expected in the pytest.raises() value.
- The subprocess.CompletedProcess object has a returncode attribute set to 1.
- The subprocess.CompletedProcess object has an stdout attribute that contains the string 'exit 1'.
- The subprocess.CompletedProcess object has an stderr attribute that contains the string 'Authentication failed'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: The test verifies that the TokenRefresher raises an error when given an empty output.

Why Needed: This test prevents a potential bug where the TokenRefresher does not raise an error for an empty output, potentially leading to unexpected behavior or silent failures.

Key Assertions:

- the function `get_token()` returns an empty string as its stdout.
- the function `get_token()` raises a `TokenRefreshError` exception with the message 'empty output'.
- the error message is not truncated to only contain 'empty output', allowing for silent failures.

COVERAGE

src/pytest_llm_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 `force_refresh` bypasses cache and returns a new token when called with `True`.

Why Needed: This test prevents a potential bug where the cache is not cleared when `refresh_interval` is set to a value greater than 1 hour, causing subsequent calls to return cached tokens instead of refreshing them.

Key Assertions:

- The function `get_token()` returns a new token with the correct name (`token-2`) after calling `force=True`.
- The function `get_token()` clears the cache and sets `call_count` to 2 when called without any arguments.
- The function `get_token()` does not return cached tokens when `refresh_interval` is set to a value greater than 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/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 accessing an access token.

Why Needed: This test prevents a potential bug where the `TokenRefresher` class does not handle cases where the custom JSON key is missing or invalid.

Key Assertions:

- The `get_token()` method of the `TokenRefresher` instance should return the custom JSON key.
- The `json_key` parameter passed to the `TokenRefresher` constructor should be set to 'access_token'.
- The `output_format` parameter passed to the `TokenRefresher` constructor is set to 'json'.
- The `command` parameter passed to the `TokenRefresher` constructor is set to 'get-token'.
- The `refresh_interval` parameter passed to the `TokenRefresher` constructor is set to 3600 seconds.
- The `output_format` parameter is set to 'json' and the `json_key` parameter is set to 'access_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/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

Scenario: Verify that the `TokenRefresher` extracts a 'token' key from the JSON output of the `get-token` command.

Why Needed: This test prevents a potential bug where the `TokenRefresher` does not extract the expected 'token' key from the JSON output, potentially leading to incorrect token usage or storage.

Key Assertions:

- The 'token' key should be present in the JSON output.
- The value of the 'token' key should match the expected string value.
- The 'expires_in' key should not be present in the JSON output if it is not a valid token refresh interval.
- The 'command' key should be present in the JSON output and contain the correct command to get the token.
- The 'refresh_interval' key should be present in the JSON output and match the expected value (3600 seconds).
- The 'output_format' key should be present in the JSON output and match the expected value ('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` correctly extracts the text format of a token from its output.

Why Needed: This test prevents a potential bug where the `get_token_text_format` method does not extract the correct text format of the token.

Key Assertions:

- The output of the `get-token` command is 'INFO: Processing...my-secret-token' which indicates that the text format is 'text'.
- The extracted text should be exactly 'my-secret-token'.
- The output does not contain any non-text characters.
- The output does not contain any whitespace characters after the token name.
- The output does not contain any newline characters at the end of the string.
- The output is a single line with no additional information.
- The output does not have any special formatting or encoding that could affect the text extraction.

COVERAGE

src/pytest_llm_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 TokenRefresher raises an error when it encounters invalid JSON.

Why Needed: This test prevents the introduction of a bug where the TokenRefresher class incorrectly handles or interprets invalid JSON input.

Key Assertions:

- The `get_token()` method of the `TokenRefresher` instance should raise a `TokenRefreshError` exception when it encounters an invalid JSON string.
- The error message returned by the `get_token()` method should indicate that the input is not valid JSON.
- The test should fail with a `TokenRefreshError` exception when the `get_token()` method is called with an invalid JSON string.
- The `json` key in the error message should be present and contain the word 'json'.
- The test should only pass if the input JSON string is not valid, i.e., it does not start with a double quote or end with a backslash.
- The `json` key in the error message should also contain the word 'json' after the colon.
- The test should fail when the input JSON string contains an array of objects instead of a single object.
- The `json` key in the error message should indicate that the input is not valid JSON, even if it contains multiple objects.
- The test should only pass if the input JSON string does not contain any quotes or backslashes.

COVERAGE

src/pytest_llm_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: The `invalidate` method of the `TokenRefresher` class should clear the cache after a refresh operation.

Why Needed: This test prevents a potential bug where the cache is not cleared after a refresh, potentially leading to stale token data.

Key Assertions:

- token1 was set with the correct value before the invalidate call
- token2 was set with the correct value after the invalidate call
- call_count was incremented correctly during the fake_run function
- the output of `subprocess.run` matches the expected token values
- the cache is cleared and refresher.invalidate() returns a CompletedProcess object with returncode 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	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 an error when the JSON key is missing from the token response.

Why Needed: This test prevents a potential bug where the TokenRefresher class does not raise an exception when the required JSON key is missing from the token response.

Key Assertions:

- The 'token' key should be present in the token response.
- The 'not found' message should be returned when the 'token' key is missing.
- The 'other_key' value should not be present in the token response.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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 potential threading issues where multiple threads may attempt to acquire the lock at the same time, leading to inconsistent results or errors.

Key Assertions:

- all threads should get the same token (first one to acquire lock)
- the first thread to acquire the lock should retrieve the token 'token-1'
- subsequent threads should not interfere with each other's token retrieval
- no other processes should be able to retrieve a different token while another thread is still holding the lock
- the output of all threads should match (i.e., they will all print 'token-X' where X is their respective tokens)
- all results from get_token() calls should be equal

COVERAGE

src/pytest_llm_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 correctly handles command timeouts.

Why Needed: This test prevents the TokenRefresher from crashing unexpectedly when a timeout occurs.

Key Assertions:

- The `get_token()` method raises a `TokenRefreshError` with the message 'timed out'.
- The error message contains the string 'timed out' in lowercase.
- The `pytest.raises(TokenRefreshError)` assertion checks for the presence of the 'timed out' error message.
- The `assert` statement checks that the error message is present and contains the string 'timed out'.
- The test verifies that the error message does not contain any other relevant information.
- The test ensures that the error message is correctly formatted and includes the necessary details.
- The test provides a clear indication of what went wrong when a timeout occurs.

COVERAGE

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

Why Needed: Prevents potential infinite command calls due to caching.

Key Assertions:

- The token is retrieved from the cache and matches the expected value.
- The same token is cached again, preventing unnecessary command calls.
- Only one call to the `get-token` method is made, as expected by the caching mechanism.

COVERAGE

src/pytest_llm_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

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	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 not properly handled, potentially leading to unexpected behavior or errors.

Key Assertions:

- The function `get_token()` raises a `TokenRefreshError` with the message 'empty'
- The error message includes the word 'empty' in lowercase
- A test assertion verifies that the error message contains the string '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) when the provided command contains shell syntax that is not supported by TokenRefresher.

Why Needed: To prevent a potential bug where the token refresh fails due to an invalid command string containing unescaped quotes or other shell-specific characters.

Key Assertions:

- The `TokenRefreshError` exception is raised with a message indicating that the provided command contains an invalid syntax.
- The error message includes the phrase 'Invalid command string'.
- The test asserts that the error message contains the specified phrase to verify its correctness.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 is not a dictionary.

Why Needed: This test prevents a potential bug where the TokenRefresher incorrectly handles JSON output as a list instead of a dictionary.

Key Assertions:

- The function `get_token()` should raise a `TokenRefreshError` with an error message indicating that the output is not a dictionary.
- The error message should include the string 'Expected JSON object'.
- The error message should include the string '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/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 unexpected behavior when the token value is an empty string, which could cause errors during refresh.

Key Assertions:

- The function `get_token()` should raise a `TokenRefreshError` with the message 'empty or not a string' when the input token value is an empty string.
- The function `get_token()` should return a `ValueError` with the message 'empty or not a string' when the input token value is an empty string.
- The function `get_token()` should raise a `TypeError` when the input token value is not a string.
- The function `get_token()` should check if the input token value is an empty string before attempting to process it.
- The function `get_token()` should handle the case where the token value contains only whitespace characters (e.g., ' ').
- The function `get_token()` should not attempt to parse or validate the token value when it's an empty string.
- The function `get_token()` should return a meaningful error message when the input token 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/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` 4ms 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	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 the TokenRefresher's handling of OSError when executing a command.

Why Needed: This test prevents a potential regression where the TokenRefresher fails to handle OSError when trying to refresh tokens.

Key Assertions:

- The function `get_token()` should raise a `TokenRefreshError` with a message indicating that the command was not found.
- The error message should contain the string 'Failed to execute'.
- The `exc_info.value` attribute should be an instance of `OSError`.

COVERAGE

src/pytest_llm_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. This test verifies that the TokenRefresher correctly handles cases where the output of a command has only blank lines (i.e., whitespace) after the initial strip is done.

Why Needed: This test prevents regression in coverage of edge cases where the output of a command may have only whitespace lines after the initial strip is done, which could lead to unexpected behavior or errors if not handled properly.

Key Assertions:

- The _parse_output method should raise a TokenRefreshError when it encounters only whitespace lines after the initial strip.
- The parsed output should contain non-empty lines but have only whitespace content lines.
- The error message should indicate that there are no non-empty lines in the output.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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 that the TokenRefresher raises a TokenRefreshError when given an empty command string.

Why Needed: To prevent a potential issue where an empty command string is passed to the TokenRefresher, causing it to raise a TokenRefreshError.

Key Assertions:

- The get_token() method of the TokenRefresher should return a TokenRefreshError with the message 'empty' when given an empty command string.
- The error message returned by the get_token() method should be case-insensitive (i.e., 'Empty' instead of 'EMPTY').
- The error message returned by the get_token() method should not contain any whitespace characters.
- The get_token() method should raise a TokenRefreshError with the specified error message when given an empty command string.
- The error message returned by the get_token() method should be raised as a pytest.raises() 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/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)