Test Report

Run ID: 21101927020-py3.12 • Generated: 2026-01-17 22:36:57 • Duration: 16.81s
Plugin: v0.1.0 (b7a157f6cb9189cc50a17c846484c8454deeac61) [dirty]
Repo: v0.1.1 (17ba445b18c6b8e678634e3310a2851e05f74129)
LLM: ollama / llama3.2:1b (minimal context, 366 annotated, 36 errors)

90.11%
Total Coverage

403	403	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	10	125	92.59%	13, 15-17, 19-22, 30, 38, 44-45, 47, 58-60, 64, 72-73, 83, 86, 88-90, 92, 94-96, 98, 101-104, 106-108, 114, 116, 118, 121-122, 127, 131-135, 137-140, 144-146, 148, 150, 152-153, 156, 160-162, 165, 167-168, 173, 176, 178-184, 187-189, 191, 196, 199-200, 202, 204,	62, 123, 125, 128, 157, 221, 249, 251, 257, 274

					216-217, 220, 224-225, 228-234, 236, 239, 241, 243-244, 246-248, 250, 252-254, 259-260, 263-264, 271, 273, 276-279, 281-283, 285, 299-300, 302, 308	
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	8	54	87.1%	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, 142, 164, 175-176, 179-181, 183, 185-186, 188,	123-124, 126, 133, 135-136, 138, 191

					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	7	64	90.14%	7, 9-14, 17, 20, 23-24, 36-39, 41-43, 47, 59-60, 63-66, 69-72, 74, 83, 85-86, 90, 93, 101-103, 107-109, 111, 113-115, 120, 132-135, 139-140, 143-144, 148-150, 153-154, 156, 158, 160-162	87-88, 91, 116, 136, 145, 155
src/pytest_llm_report/models.py	243	10	233	95.88%	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, 173-177, 179-188, 190, 192, 194-196, 199-200, 208-209, 211, 213, 219-220, 229-231, 233, 235, 239-241, 244-245, 254-256, 258, 260, 267-268, 277-279,	178, 189, 191, 193, 466, 519, 521, 523, 525, 527

					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, 514-518, 520, 522, 524, 526, 528	
src/pytest_llm_report/options.py	138	55	83	60.14%	122, 162, 191, 194-196, 201-203, 209-211, 217-219, 225-226, 233, 237-246, 248, 252, 261, 276, 279-295, 298, 305-308, 310-312, 319-332, 335-344, 347, 349	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, 227, 234, 250, 255, 264, 315-316
src/pytest_llm_report/plugin.py	157	24	133	84.71%	40, 43, 49, 55, 61, 67, 73, 80, 89, 95, 101, 107, 113, 121, 126, 131, 136, 142, 147, 153, 160, 165, 170, 175, 181, 186, 202, 206, 210, 216-217, 220-221, 223, 225, 228-230, 236-237, 245-246, 271-272, 275-276, 279, 282-283, 285-286, 289-290, 292, 294-298, 301-302, 304, 306, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-338, 340, 342-345, 348-349, 357-358, 363-366, 369, 371, 374-379, 381, 383, 391-392, 413-414, 417-418, 421-423, 434-435, 438, 441-442, 445-447, 457-458, 461-463,	13, 15-17, 19-20, 22, 28-31, 34, 193, 249, 353-354, 359-360, 405-406, 426, 450, 466-467

					474-475, 478, 481, 483-484	
src/pytest_llm_report/prompts.py	75	5	70	93.33%	13, 15-17, 24, 27, 33, 35, 49, 52, 55, 58-61, 63, 65, 67, 78-79, 82-84, 86-87, 92, 94-95, 98-101, 103-112, 116, 118, 132-133, 135-138, 140-141, 144-145, 148, 151-152, 154-156, 158-159, 163, 165, 180, 182, 191-194	80, 114, 142, 146, 149
src/pytest_llm_report/render.py	50	0	50	100.0%	13, 15-16, 18, 24, 30-31, 34, 40, 42, 50-51, 53, 56, 65-67, 70, 79, 87, 90, 99, 101-102, 107, 110, 121-124, 126-129, 131-134, 141-143, 145, 158-163, 177, 196	-
src/pytest_llm_report/report_writer.py	167	10	157	94.01%	13, 15-25, 27-29, 46, 55, 58, 67-68, 76, 83-84, 89, 98-100, 102, 105-108, 110, 116, 127-128, 130, 142, 150, 156-158, 160, 186-189, 192, 197-199, 202-203, 211, 222-223, 226-227, 230-231, 233, 235, 254, 256-259, 262-264, 266, 268, 303, 312, 314-315, 317-328, 330, 332, 340, 343-345, 348-349, 352-354, 357, 360, 368, 376, 378-379, 382, 385, 388, 391, 399, 401-402, 408, 410, 412, 414-423, 434-435, 437-439, 447-448, 453, 455, 458,	113, 135-137, 424-425, 432, 449-451

					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	3	31	91.18%	11, 13-14, 17, 30, 33, 36, 39, 42, 45, 55-56, 58-60, 63-64, 70, 79, 82, 100, 103, 111-113, 116-117, 119-121, 123	40, 65, 67
src/pytest_llm_report/util/hashing.py	36	0	36	100.0%	12, 14-17, 23, 32, 35, 44-48, 51, 61, 64, 73-74, 76-78, 80-81, 86, 96, 103-104, 107, 113-114, 116-121	-
src/pytest_llm_report/util/ranges.py	33	0	33	100.0%	12, 15, 29-30, 33, 35-37, 39-40, 42, 45-47, 50, 52, 55, 65-67, 70, 81-82, 84-91, 93, 95	-
src/pytest_llm_report/util/time.py	16	0	16	100.0%	4, 6, 9, 15, 18, 27, 30, 39-44, 46-48	-

📄 **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 results from multiple runs.

Why Needed: Prevents regression in aggregate policy when running multiple tests concurrently.

Key Assertions:

- The aggregated report contains both retained test cases.
- The number of retained test cases is consistent across all runs.
- The aggregated report does not contain any duplicate test case IDs.
- All retained test cases are present in the final aggregated result.
- No test cases are skipped or lost during aggregation process.
- The aggregated report includes all specified test cases and their outcomes.
- The aggregated report is consistent across different run numbers.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verifies that the aggregate function does not attempt to aggregate a directory that does not exist.

Why Needed: Prevents a potential bug where the aggregate function throws an exception when trying to aggregate a non-existent directory.

Key Assertions:

- The `aggregate` method is called with an empty list.
- A `PathError` is raised with a message indicating that the specified path does not exist.
- The `aggregate` method returns `None` as expected.
- The test passes even if the directory exists in the current working directory.
- The test fails when the directory exists in a different location.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the latest policy is selected when aggregating reports with different times but same test outcome.

Why Needed: This test prevents regression where the latest policy might not be chosen due to inconsistent report timings.

Key Assertions:

- The result of `aggregate()` should contain only one test.
- The first test in the result list should have an 'outcome' of 'passed'.
- The aggregated run meta should indicate that all tests ran.
- All tests passed and no failed tests should be reported.
- The summary should contain 1 passed test and 0 failed tests.

COVERAGE

src/pytest_llm_report/aggregation.py	78 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 130-131, 134, 141, 147, 149, 151-156, 158, 160-162, 173, 185, 187-191, 193-194, 197, 220, 222-226, 238, 248, 251-252, 254, 256, 279-282, 284)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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 would attempt to aggregate data without a configured aggregation directory.

Key Assertions:

- The `aggregate()` method of the `Aggregator` instance should return `None` when called with an empty or None `aggregate_dir` attribute.
- An exception should not be raised if the `aggregate_dir` attribute is set to `None`.
- The aggregate function should not attempt to aggregate data without a configured aggregation directory.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The `aggregate` method of the Aggregator instance should be called when no reports are available.

Why Needed: This test prevents a potential bug where the `aggregate` method does not get called with an empty list of reports, potentially leading to incorrect aggregation results or errors.

Key Assertions:

- The `aggregate` method is called with an empty list of reports.
- An empty list of reports is returned from the `aggregate` method.
- The `aggregate` method does not raise an exception when no reports are available.
- The `aggregate` method does not log any error messages when no reports are available.
- The `aggregate` method does not update the aggregation result with an empty list of reports.
- The `aggregate` method returns a None value instead of raising an exception when no reports are available.
- The Aggregator instance is updated correctly after calling the `aggregate` method without any reports.

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: 413-414, 417, 421-423)

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: 413-414, 417, 421-423)

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 aggregation functionality where source coverage data is expected to be deserialized.

Key Assertions:

- The 'source_coverage' key should exist in the aggregated report.
- The 'source_coverage' value should be a list of SourceCoverageEntry objects.
- Each SourceCoverageEntry object should have the required keys (file_path, statements, missed, covered, coverage_percent, covered_ranges, missed_ranges).
- The file path of each SourceCoverageEntry should match the provided source code file.
- The number of statements in the aggregated report should be equal to the sum of statements in all individual reports.
- Each statement should have a corresponding range (e.g., 1-5, 7-11).
- All coverage percentages should add up to 100%.
- All covered ranges should match the provided ranges.
- All missed ranges should be empty strings.
- The aggregated report should not be None.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents a bug where the aggregator tries to load coverage without setting the llm_coverage_source configuration option.

Key Assertions:

- Verify that _load_coverage_from_source() returns None when llm_coverage_source is set to None.
- Verify that _load_coverage_from_source() raises UserWarning with message 'Coverage source not found' when llm_coverage_source is set to '/nonexistent/coverage'.
- Verify that _load_coverage_from_source() returns a mock coverage object (80.0) when llm_coverage_source is set to '.coverage'.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the `_recalculate_summary` method to ensure it recalculates the latest summary correctly when new test results are added.

Why Needed: This test prevents regression in the `aggregator._recalculate_summary` method, which is responsible for updating the latest summary with new test results.

Key Assertions:

- The total count of tests passed should be equal to the initial total count.
- The number of failed tests should remain unchanged.
- The number of skipped tests should also remain unchanged.
- The number of tests that failed or were skipped but not yet counted should still be 1.
- The number of tests that have been counted (passed, failed, and skipped) should still be 6.
- The coverage percentage should still be preserved at 85.5%.
- The total duration of the latest summary should remain unchanged at 5.0 seconds.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test verifies that the test_skips_invalid_json function skips an invalid JSON file.

Why Needed: This test prevents a regression where the test SkipsInvalidJson function fails to skip an invalid JSON file due to missing required fields in the report.

Key Assertions:

- The function should not count the invalid JSON file as a valid report.
- The function should raise a UserWarning with the message 'Skipping invalid report file' when encountering an invalid JSON file.
- The function should only count the valid report as a valid report.
- The function should ignore the missing fields in the invalid JSON file and not include it in the aggregation result.
- The function should return None for the aggregation result after skipping the invalid JSON file.

COVERAGE

src/pytest_llm_report/aggregation.py	71 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119-120, 125, 127-128, 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: 413-414, 417, 421-423)

📄 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 `_recalculate_summary` method returns a Summary object with the correct total, passed, and failed counts.

Why Needed: This test prevents regression where the coverage percentage is not calculated correctly due to missing or incomplete tests.

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: 413-414, 417, 421-423)

📄 tests/test_annotator.py

7 tests

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

AI ASSESSMENT

Scenario: The test verifies that cached tests are skipped.

Why Needed: This test prevents a potential performance regression where cached tests are not properly skipped.

Key Assertions:

- mock_provider is called with the same arguments as when no caching is enabled
- mock_cache is not called in the test
- mock_assembler is not called in the test

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-81, 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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Testing concurrent annotation of a test function

Why Needed: Prevents potential performance issues or data corruption caused by concurrent access to the annotator.

Key Assertions:

- The `test_concurrent_annotation` method is called with mock providers, cache and assembler instances.
- Each mock provider and cache instance is not reused across multiple calls.
- The annotator does not modify any external state that could be affected by concurrent access.
- No shared data structures are accessed or modified by the annotator.
- All mock providers, cache and assembler instances are properly cleaned up after each test call.
- No exceptions are raised when calling `test_concurrent_annotation` method.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The annotator handles failures concurrently without any noticeable impact on performance or correctness.

Why Needed: This test prevents a potential regression where the annotator fails to correctly handle cases where multiple annotations are processed simultaneously and fail.

Key Assertions:

- mock_assembler should not raise an exception when processing annotations in parallel.
- mock_provider should return a failure response for each annotation that fails to be annotated.
- mock_cache should not be modified by the annotator during concurrent annotation.
- the annotator's output should not contain any information about failed annotations.
- the annotator's error messages should not indicate that multiple annotations were processed simultaneously and failed.
- the annotator's performance metrics (e.g., annotation rate) should remain unchanged even when handling failures concurrently.
- the test should be able to reproduce the failure scenario reliably, without requiring additional setup or configuration.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The `test_progress_reporting` function is used to verify the annotator's progress reporting capabilities.

Why Needed: This test prevents regressions where the annotator fails to report progress correctly after a large number of annotations.

Key Assertions:

- mock_provider.get_progress() should return a non-negative value
- mock_cache.get_annotations() should be called with the correct arguments
- mock_assembler.get_annotation() should be called with the correct annotation ID

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The `test_sequential_annotation` function is being tested to ensure it correctly annotates sequential data.

Why Needed: This test prevents a potential regression where the sequential annotation might not work as expected due to incorrect handling of asynchronous data.

Key Assertions:

- Mocking the `mock_provider`, `mock_cache`, and `mock_assembler` objects ensures that they are properly mocked for the purpose of testing the sequential annotation.
- The test verifies that the annotated sequential data is correctly generated without any errors or exceptions.
- The test checks that the annotator function can handle asynchronous data correctly by verifying that it does not raise any exceptions.
- The test ensures that the `mock_assembler` object is properly mocked to simulate the expected behavior of an assembler.
- The test verifies that the annotated sequential data is consistent with the original data before and after annotation.
- The test checks that the annotator function can handle edge cases such as empty or null data correctly.
- The test ensures that the `mock_provider` object is properly mocked to simulate the expected behavior of a provider.
- The test verifies that the annotated sequential data is correctly generated without any errors or exceptions.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the annotator would skip tests or annotations if the LLM was enabled, potentially causing unintended behavior in downstream processes.

Key Assertions:

- The function `annotate_tests` does not modify any configuration or perform any actions when the LLM is disabled.
- The `config` object passed to `annotate_tests` has a 'provider' key with value 'none'.
- The `annotate_tests` function does not attempt to annotate any tests or annotations.
- The `None` provider is used, which indicates that no external data source is being utilized.
- No configuration changes are made when the LLM is disabled.
- The annotator remains silent and does not perform any actions.
- The test results do not indicate any issues with the annotator's behavior.
- The `annotate_tests` function behaves as expected when the LLM is disabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents regressions where providers are not available and the annotator still attempts to annotate.

Key Assertions:

- Mocked `provider` object should be mocked with an `unavailable` status.
- Captured output from `sys.stderr` should indicate that the provider is unavailable.
- The `annotator` function should skip the annotation process when a provider is unavailable.
- No exceptions should be raised in this case, indicating successful annotation.
- The annotator's progress should not be affected by the provider being unavailable.
- The annotator's output should still contain the expected annotations even if the provider is unavailable.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

📄 tests/test_annotator_maximal.py

4 tests

PASSED `tests/test_annotator_maximal.py::TestAnnotatorAdvanced::test_annotate_concurrent_with_progress_and_errors` 2ms 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 in concurrent mode where LLM annotations are reported with progress and errors.

Key Assertions:

- The function should report the correct number of annotated tasks (2) and failures (1).
- The first error message should contain the string 'first error'.
- At least one processing message should be generated for each task, containing the string 'LLM annotation'.
- All progress messages should contain the string 'Processing X test(s)', where X is the number of tasks.
- The function should correctly handle cases where there are more or fewer tasks than specified in the config (2 vs 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/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: 413-414, 417, 421-423)`

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

AI ASSESSMENT

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

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

Key Assertions:

- The time.sleep() function was called before the LlmAnnotation task finished.
- The time.sleep() function was called after the LlmAnnotation task finished but within the specified rate limit interval of 1.0s.
- The time.sleep() function was not called at all if only 0.1s elapsed but the rate limit interval is 1.0s.
- The mock_time.side_effect was set to [100.0, 100.1, 100.2, 100.3, 100.4] which indicates that time.sleep() function has been called multiple times with different values.
- The LlmAnnotation task finished after the rate limit interval of 1.0s but before the mock_time.side_effect was set to [100.0, 100.1, 100.2, 100.3, 100.4].
- The LlmAnnotation task did not finish within the specified rate limit interval of 1.0s.
- The time.sleep() function is called before the LlmAnnotation task starts or after it finishes but within the specified rate limit interval of 1.0s.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	23 lines (ranges: 165-168, 170-171, 173-174, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Should report progress for cached tests when annotating tests with maximal caching.

Why Needed: This test prevents regression where the annotator does not report progress for cached tests, potentially leading to missed opportunities for optimization or improvement.

Key Assertions:

- The `progress_msgs` list should contain any string that starts with '(cache): '
- Each item in `progress_msgs` should be a string that contains the scenario 'test_cached'
- All strings in `progress_msgs` should not be empty
- There should be at least one string in `progress_msgs` that matches '(cache): test_cached'
- The last item in `progress_msgs` should contain the scenario 'test_cached'

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that when the provider is not available, it prints a message and returns without attempting to annotate tests.

Why Needed: This test prevents regression or bug in the annotator's behavior when the provider is unavailable.

Key Assertions:

- mocks.get_provider().is_available() was called with False
- the annotated tests are skipped
- the message 'not available. Skipping annotations' is printed to stdout

COVERAGE

src/pytest_llm_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: 413-414, 417, 421-423)

📄 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: Test that an error is thrown when parsing a malformed JSON after extracting from the response.

Why Needed: To prevent a JSONDecodeError when the extracted JSON does not match the expected format.

Key Assertions:

- The `annotation.error` attribute should contain 'Failed to parse LLM response as JSON'.
- The `provider._parse_response(response)` call should raise a `JSONDecodeError`.
- The error message should be 'Failed to parse LLM response as JSON'.
- The extracted JSON is invalid and cannot be parsed by the LLM.
- The `extract_json_from_response` method finds braces, but the contents are malformed.
- The expected format of the extracted JSON does not match the actual 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Tests the _parse_response method with non-string fields to ensure it correctly handles scenario/why_needed.

Why Needed: This test prevents a potential bug where the _parse_response method incorrectly interprets non-string fields as lists, leading to incorrect results or errors.

Key Assertions:

- annotation.scenario = '123' matches the expected value
- annotation.why_needed = ['list'] matches the expected list of why needed scenarios
- annotation.key_assertions = ['a'] matches one of the key assertions in the response data

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	22 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203-207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

📄 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 issue where an invalid or missing configuration is passed to the `get_gemini_provider` function, potentially causing it to return an incorrect provider.

Key Assertions:

- The returned value is an instance of `GeminiProvider`.
- The returned value has the correct class name (`GeminiProvider`).
- The returned value does not have any attributes or methods that are not present in `GeminiProvider`.
- The returned value's `__class__` attribute matches the expected value (`GeminiProvider`).
- The provided configuration is valid and does not contain any invalid or missing settings.
- No exceptions are raised when calling `get_gemini_provider` 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents the introduction of a bug where an unknown LLM provider is used without validation.

Key Assertions:

- The function `get_provider` raises a `ValueError` with a message indicating that the provided LLM provider is unknown.
- The error message includes the string 'invalid' to identify the invalid provider.
- The test verifies that the `pytest.raises` context manager is used correctly to catch the ValueError exception.
- The test checks that the `match` parameter of the `pytest.raises` context manager matches the expected error message.
- The test ensures that the error message includes the string 'Unknown LLM provider: invalid'.
- The test verifies that the `Config` class is used correctly to specify an invalid provider.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

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`.

Why Needed: Prevents a potential bug where the test fails if the 'litellm' provider is not installed or configured correctly.

Key Assertions:

- The returned value should be an instance of `LiteLLMProvider`.
- The returned value should have a type hint of `LiteLLMProvider`.
- The function name `get_litellm_provider` should return an instance of `LiteLLMProvider`.
- The function does not raise any exceptions if the 'litellm' provider is not installed or configured correctly.
- The function returns an instance of `LiteLLMProvider` even when the 'litellm' provider is not found in the configuration.
- The function name `get_litellm_provider` should be able to handle different providers, including 'litellm'.
- The function does not throw any errors if the 'litellm' provider is not installed or configured correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verifies that the `get_noop_provider` function returns an instance of `NoopProvider` when no provider is specified.

Why Needed: Prevents a potential bug where the test fails if no provider is specified in the configuration.

Key Assertions:

- The `get_provider` function should return an instance of `NoopProvider`.
- The `provider` attribute of the returned `NoopProvider` instance should be `None`.
- The `Config` class constructor with a `provider` parameter should not throw an exception if no provider is specified.
- The `get_provider` function should handle cases where no provider is specified in the configuration without raising an error.
- The `provider` attribute of the returned `NoopProvider` instance should be `None` when no provider is specified.
- The `Config` class constructor with a `provider` parameter should not throw an exception if no provider is specified.
- The `get_provider` function should return an instance of `NoopProvider` when no provider is specified 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	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: 413-414, 417, 421-423)

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` for a valid configuration.

Why Needed: This test prevents a potential bug where the `get_ollama_provider` function does not return an instance of `OllamaProvider` when a valid configuration is provided.

Key Assertions:

- The `provider` attribute of the returned `OllamaProvider` instance is set to `'ollama'`.
- The `provider` attribute of the returned `OllamaProvider` instance is not set to an empty string or None.
- The `provider` attribute of the returned `OllamaProvider` instance has a value that matches the expected configuration ('ollama').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 245, 247, 249, 252-253, 255)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that the `is_available()` method returns a boolean indicating availability

Why Needed: This test prevents regression in cases where LLM providers are not available due to cache issues.

Key Assertions:

- The `is_available()` method should return True when the provider is configured correctly and has no cached results.
- The `is_available()` method should return False when the provider is configured incorrectly or has cached results.
- The `checks` attribute of the provider should be incremented each time `_check_availability()` is called.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 107-108, 110-111)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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 model name specified in the configuration.

Why Needed: This test prevents a regression where the default model name is not correctly set to the configuration.

Key Assertions:

- assert provider.get_model_name() == 'test-model'
- provider.config.model == 'test-model'
- provider.config.model_type == ModelConfig
- provider.model_name == 'test-model'
- provider.model_type == ModelType
- provider.config.name == 'TestModel'

COVERAGE

src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The `get_rate_limits` method of the `ConcreteProvider` class returns `None` when called with default configuration.

Why Needed: This test prevents a potential bug where the rate limits are not set correctly in the default configuration.

Key Assertions:

- The `rate_limit` attribute is missing from the provider's configuration.
- The `max_rate` attribute is set to `None` or an invalid value.
- The `min_rate` attribute is set to a positive value.
- The `timeout` attribute is not set.
- The `batch_size` attribute is not set.
- The `num_threads` attribute is not set.
- The `max_queue_size` attribute is not set.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verifies that the `is_local` method returns `False` when provided with a non-local configuration.

Why Needed: Prevents regression in case of incorrect or outdated configuration settings.

Key Assertions:

- The `provider.is_local()` method should return `False` for a non-local configuration.
- A non-local configuration should not trigger the `is_local` check to return `True`.
- The `is_local` method should raise an error or return a meaningful value when given an invalid configuration.
- When using a different provider, the `is_local` method should still return `False` for a non-local configuration.
- A local configuration should not be considered valid even if the `provider` is set to `None` or an empty object.
- The test should cover cases where the `config` object has been modified after the `provider` was created.

COVERAGE

src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

📄 tests/test_cache.py

7 tests

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

AI ASSESSMENT

Scenario: The function `hash_source` is called with a source code that produces the same result.

Why Needed: This test prevents a bug where different sources of input produce different hashes, potentially leading to inconsistencies in caching.

Key Assertions:

- source_code_is_same
- hashes_are_equal
- same_source_produces_same_hash

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Testing the cache with different sources and hashes.

Why Needed: This test prevents a potential issue where the same function call results in different hash values due to caching.

Key Assertions:

- The `hash_source` function should return a different hash value for two different source strings.
- The `hash_source` function should not return the same hash value for the same source string and a different function name.
- The `hash_source` function should handle cases where the function body is identical but has different names.
- The `hash_source` function should ignore case differences in the source strings.
- The `hash_source` function should not cache function calls with the same source string and different function names.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verifies the length of a hashed string.

Why Needed: Prevents a bug where the hash length is not consistent across different inputs.

Key Assertions:

- The length of the hash should be exactly 16 characters.
- The hash value for the input 'test' should have a length of 16 characters.
- The hash value for other inputs should also have a length of 16 characters.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that clearing the cache removes all entries.

Why Needed: Prevents regression where cache entries are not properly cleared after test execution.

Key Assertions:

- The number of cache entries should be reduced to 2 after clearing.
- A specific annotation ('test::a', 'hash1') should no longer exist in the cache.
- A specific annotation ('test::b', 'hash2') should no longer exist in the cache.
- All cache keys for a given scenario should not have any entries.
- The value associated with an entry of a given scenario and key should be None after clearing.

COVERAGE

src/pytest_llm_report/cache.py	26 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 129, 132-136, 141)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

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: 413-414, 417, 421-423)

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 a missing entry in the cache causes an incorrect result when trying to access it.

Key Assertions:

- The `get` method should return `None` when called with a non-existent key.
- The `cache.get()` function should raise a `KeyError` exception when given a non-existent key.
- The `result` variable should be set to `None` after calling `cache.get()`.
- The test should fail when the `get` method returns `None` for a missing entry in the cache.
- The test should verify that an error is raised when trying to access a non-existent key in the cache.

COVERAGE

src/pytest_llm_report/cache.py	9 lines (ranges: 39-41, 53, 55-56, 118-119, 121)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: Prevents bypass by ensuring annotations are cached before they are used to generate the response.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/cache.py	28 lines (ranges: 39-41, 53, 55, 58, 60-62, 68-73, 86, 90, 92, 94, 97-101, 103, 118-119, 121)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

📄 tests/test_collector.py

11 tests

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

AI ASSESSMENT

Scenario: Test verifies that collection errors have the correct structure.

Why Needed: This test prevents a potential bug where incorrect structure is reported for collection errors.

Key Assertions:

- The nodeid of the error should match the expected value 'test_bad.py'.
- The message of the error should be '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: 413-414, 417, 421-423)

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 an empty collection is returned unexpectedly.

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 handles the case of 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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the default value of llm_context_override in TestCollectorMarkerExtraction.

Why Needed: Prevents a potential bug where the default value of llm_context_override is not set correctly.

Key Assertions:

- The llm_context_override attribute should be None for the test case.
- The TestCaseResult object should have an llm_context_override attribute with a value of None.
- If llm_context_override is not None, it should be set to None in the TestCaseResult object.
- If llm_context_override is not None, it should be set to None before the test passes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test case "tests/test_collector.py::TestCollectorMarkerExtraction::test_llm_opt_out_default_false" verifies that the default value of llm_opt_out is set to False.

Why Needed: This test prevents a regression where the default value of llm_opt_out was incorrectly set to True, potentially causing issues with the test report.

Key Assertions:

- The llm_opt_out attribute should be set to False.
- The llm_opt_out attribute should not be set to True.
- The llm_opt_out attribute should have a default value of False.
- The TestCaseResult object should contain an llm_opt_out attribute with a value of False.
- The TestCaseResult object should contain an llm_opt_out attribute that is not 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/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

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

AI ASSESSMENT

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

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

Key Assertions:

- config.capture_failed_output should be set to False
- the `capture` feature should not be enabled by default
- output capture functionality should only be available when explicitly enabled
- the `capture` feature should have a clear and consistent behavior

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_collector.py::TestCollectorOutputCapture::test_capture_max_chars_default` 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	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression by ensuring that XFAILs are properly recorded and handled by the TestCollector.

Key Assertions:

- The 'xfail passes' message should be displayed as 'xpassed' when an XFAIL is passed.
- The 'xfail passes' message should not be displayed as 'failed' when an XFAIL is passed.
- The 'xfail passes' message should not be displayed as 'skipped' when an XFAIL is passed.
- The test results should correctly identify the nodeid as 'test_xfail.py::test_unexpected_pass'.
- The duration of the runtest log report should be set to 0.01 seconds.
- The longrepr field should be empty.
- The wasxfail field should contain the expected failure message.
- The outcome of the test result should be 'xpassed' when an XFAIL is passed.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `TestCollector` instance is not initialized properly, leading to incorrect results in subsequent methods.

Key Assertions:

- The `results` attribute of the `collector` object should be an empty dictionary.
- The `collection_errors` list should be an empty list.
- The `collected_count` attribute of the `collector` object should be 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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the `get_results` method returns sorted results by node ID.

Why Needed: This test prevents a regression where the order of test results is not preserved.

Key Assertions:

- The list of node IDs returned by the `get_results` method should be in ascending order.
- The list of node IDs returned by the `get_results` method should contain all nodes from both tests.
- All nodes with a 'passed' outcome should appear before any nodes with an 'unpassed' outcome.
- If there are multiple results for the same test, they should be sorted alphabetically by node ID.
- The order of the first two tests in the list should match their original order.
- The last test in the list should not have a 'passed' or 'unpassed' outcome.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the TestCollector's handle_collection_finish method to ensure it correctly tracks collected and deselected counts.

Why Needed: This test prevents a potential bug where the count of collected items is not incremented correctly when the collector finishes collecting.

Key Assertions:

- The 'collected_count' attribute should be set to the number of collected items.
- The 'deselected_count' attribute should be set to the number of deselected items.
- The 'collected_count' and 'deselected_count' attributes should match the actual counts after calling handle_collection_finish.

COVERAGE

`src/pytest_llm_report/collector.py`	`16 lines (ranges: 78-79, 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 123, 163)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

📄 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: Verify that the test does not capture output when config is disabled and handle_report integration.

Why Needed: To prevent capturing of output in cases where the config is disabled, allowing for better integration with handle_runtest_logreport.

Key Assertions:

- The `collector.handle_runtest_logreport(report)` call should not modify the captured stdout.
- The `results` dictionary should still contain the original captured stdout value.
- The `collector.results['t']` attribute should return the original captured output.
- The `collector.results['t'].captured_stdout` attribute should be None.
- The `collector.handle_runtest_logreport(report)` call should not modify the `report.wasxfail` flag.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verifies that the `TestCollector` class captures stderr output correctly.

Why Needed: Prevents a bug where stderr is not captured as expected.

Key Assertions:

- The `captured_stderr` attribute of the `TestCaseResult` object is set to 'Some error'.
- The `report.capstderr` method is called with an argument of 'Some error'.
- The `collector._capture_output(result, report)` function is called with arguments that include a `report` object and a `result` object.
- The `captured_stderr` attribute of the `result` object is set to '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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `captured_stdout` attribute of the `TestCaseResult` object should contain the expected output.
- The `capstdout` attribute of the `report` object should be set to 'Some output'.
- The `capture_output` method should record the captured stdout correctly.

COVERAGE

src/pytest_llm_report/collector.py	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264-265, 268)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the collector fails to truncate output exceeding the maximum characters.

Key Assertions:

- The captured stdout length should be less than or equal to 10 (the capture_output_max_chars parameter).
- The captured stderr length should be zero, indicating no error message was truncated.
- The `captured_stdout` attribute of the `TestCaseResult` object should contain only the first 9 characters ('1234567890').

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Should extract markers from item and return the expected result with correct parameters and context.

Why Needed: This test prevents a potential bug where the collector does not correctly extract markers from an item, leading to incorrect results in subsequent steps.

Key Assertions:

- item.callspec.id should be set to 'param1' after calling get_closest_marker('llm_opt_out')
- result.param_id should be set to 'param1'
- result.llm_opt_out should be set to True after calling get_closest_marker('llm_opt_out')
- result.llm_context_override should be set to 'complete' after calling get_closest_marker('llm_context')
- result.requirements should contain exactly two strings: 'REQ-1' and '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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the `extract_error` method handles ReprFileLocation correctly and does not crash when called with a string representation.

Why Needed: To prevent a potential crash caused by using `str()` on a `ReprFileLocation` object in the `extract_error` method.

Key Assertions:

- The `extract_error` method should return 'Crash report' without crashing when called with a `Report` object containing a `longrepr` attribute.
- The `extract_error` method should not crash if it encounters an instance of `str` that is equal to the string representation of a `ReprFileLocation` object.
- The `extract_error` method should return the correct error message when called with a `Report` object containing a `longrepr` attribute.
- The `extract_error` method should not crash if it encounters an instance of `str` that is equal to the string representation of a `ReprFileLocation` object without causing a crash.
- The `extract_error` method should handle cases where the `Report` object does not contain a `longrepr` attribute, and still return 'Crash report'.
- The `extract_error` method should not crash if it encounters an instance of `str` that is equal to the string representation of a `ReprFileLocation` object without causing a crash.
- The `extract_error` method should handle cases where the `Report` object contains multiple `longrepr` attributes, and still return 'Crash report'.
- The `extract_error` method should not crash if it encounters an instance of `str` that is equal to the string representation of a `ReprFileLocation` object without causing a crash.
- The `extract_error` method should handle cases where the `Report` object contains a `longrepr` attribute with a different type than `str`, and still return 'Crash report'.

COVERAGE

src/pytest_llm_report/collector.py	22 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the `_extract_error` method returns the correct string when given a `report` object with a `longrepr` attribute.

Why Needed: This test prevents regression where an incorrect or incomplete error message is returned in cases where the error occurs before the collector has a chance to extract it.

Key Assertions:

- The `_extract_error` method should return the string provided by the `report.longrepr` attribute.
- The extracted string should match the original value of `report.longrepr` exactly.
- If an error occurs before the collector can extract it, the test should still pass without raising an assertion error.

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: 413-414, 417, 421-423)

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 method returns `None` instead of raising an exception or returning a meaningful error message when no longrepr is available.

Key Assertions:

- The method should not return `None` but rather raise an exception or return a specific value indicating that no longrepr was found.
- The method should check if the `longrepr` attribute of the report object is `None` before calling `_extract_skip_reason` and raise an exception if it is.
- The method should handle cases where the `report.longrepr` attribute is not set or is empty, and return a meaningful error message in such cases.

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: 413-414, 417, 421-423)

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` function returns 'Just skipped' as the skip reason string.

Why Needed: This test prevents a potential regression where the expected output of `_extract_skip_reason(report)` is not 'Just skipped'.

Key Assertions:

- report.longrepr == 'Just skipped'
- collector._extract_skip_reason(report) == 'Just skipped'
- report.longrepr != 'Just ignored' (this test case should have a skip reason)
- collector._extract_skip_reason(report) != 'Just ignored' (this test case should have a skip reason)
- report.longrepr != 'Just skipped and then ignored' (this test case should have two different reasons for skipping)
- collector._extract_skip_reason(report) != 'Just skipped and then ignored'

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that the `TestCollector` class can extract a skip message from a tuple containing file, line and message information.

Why Needed: This test prevents a potential bug where the `TestCollector` class does not correctly extract the skip reason from tuples containing file, line and message information.

Key Assertions:

- The expected string is present in the `report.longrepr` tuple.
- The `str()` function call on the `report.longrepr` tuple returns a value that matches the expected string.
- The extracted skip reason is not empty or None.
- The skip message does not contain any newline characters.
- The skip message does not contain any whitespace characters.
- The file name in the skip message is present and correct.
- The line number in the skip message is present and correct.
- The message in the skip reason tuple is present and correct.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that the `handle_runtest_rerun` method of `TestCollector` correctly handles reruns by updating the `rerun_count` and `final_outcome` attributes in the report.

Why Needed: This test prevents a regression where the `rerun_count` attribute is not updated correctly after a rerun, potentially leading to incorrect reporting or analysis.

Key Assertions:

- The `rerun_count` attribute of the report should be set to 1 after a rerun.
- The `final_outcome` attribute of the report should be set to 'failed' after a rerun.
- The `rerun_count` attribute should increase by 1 after each rerun.
- The `final_outcome` attribute should remain unchanged (i.e., 'passed') after all reruns.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test verifies that the TestCollector handles a run test setup failure correctly.

Why Needed: This test prevents regression in handling setup failures, ensuring that the collector logs and reports the error properly.

Key Assertions:

- The report should have an 'error' outcome.
- The phase of the test should be 'setup'.
- The error message should contain '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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test case: Handle runtest teardown failure

Why Needed: Prevents regression in scenario where teardown fails after pass.

Key Assertions:

- The `teardown` report is not recorded if the teardown fails.
- The error message 'Cleanup failed' is logged when the teardown fails.
- The outcome of the runtest is set to 'error' after a teardown failure.
- The phase of the runtest is set to 'teardown' after a teardown failure.
- The error message is not cleared from the `wasxfail` attribute of the teardown report.

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: 413-414, 417, 421-423)

📄 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: Test the Gemini model parsing edge cases for coverage boosters.

Why Needed: This test prevents regression in coverage analysis when encountering edge cases with no preferred models or all models.

Key Assertions:

- The function `provider._parse_preferred_models()` returns a list of preferred models.
- The function `provider._parse_preferred_models()` returns an empty list when the model is None.
- The function `provider._parse_preferred_models()` returns an empty list when the model is '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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents over and under token limits when recording tokens but not requests.

Why Needed: This test prevents a bug where the rate limiter allows too many tokens to be recorded without preventing subsequent requests from being blocked.

Key Assertions:

- The next_available_in() method should return a non-zero value (0) for both over and under token limits.
- The record_tokens() method should not allow more than the maximum allowed limit (100 tokens per minute).
- The limiter should correctly prevent requests from being blocked when there are insufficient tokens available.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that the `to_dict()` method returns accurate coverage percentages for SourceCoverageEntry objects.

Why Needed: This test prevents regression in coverage calculation when using models to dict variants, ensuring consistency with expected behavior.

Key Assertions:

- The 'coverage_percent' key in the resulting dictionary should match the provided value (50.0) for SourceCoverageEntry objects.
- The 'error' key in the resulting dictionary should match the provided error message ('timeout') for LlmAnnotation objects.
- The 'duration' key in the resulting dictionary should match the provided duration value (1.0) for RunMeta objects.

COVERAGE

`src/pytest_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: 413-414, 417, 421-423)`

📄 tests/test_coverage_map.py

7 tests

PASSED tests/test_coverage_map.py::TestCoverageMapper::test_create_mapper 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test 'tests/test_coverage_map.py::TestCoverageMapper::test_get_warnings' verifies that the `get_warnings` method returns a list of warnings.

Why Needed: This test prevents a potential bug where the `get_warnings` method does not return a list of warnings, potentially masking important error information.

Key Assertions:

- The `get_warnings()` method should return a list of warnings.
- The `get_warnings()` method should be able to handle any configuration or environment that may result in no warnings being generated.
- Any exceptions raised by the `get_warnings()` method should be caught and reported as an error.
- The test should fail if the `get_warnings()` method returns a non-list value, indicating a bug in the implementation.
- The test should pass if the `get_warnings()` method returns a list of warnings as expected, regardless of the configuration or 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/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: 413-414, 417, 421-423)

PASSED `tests/test_coverage_map.py::TestCoverageMapper::test_map_coverage_no_coverage_file` 2ms 5

AI ASSESSMENT

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

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

Key Assertions:

- The function should return an empty dictionary.
- The function should have at least one warning.
- The `Path.exists` mock returns False and the `glob.glob` mock returns 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that the `CoverageMapper` correctly extracts node IDs for all phases when the `include_phase` parameter is set to 'all'.

Why Needed: This test prevents a regression where the coverage map might not include all phases if the `include_phase` parameter is set to 'none' or 'partial'.

Key Assertions:

- The method `_extract_nodeid` returns the correct node ID for each phase.
- The method `_extract_nodeid` returns the same node ID for multiple phases in a single test.
- The method `_extract_nodeid` handles cases where the phase name is not present in the code snippet.
- The method `_extract_nodeid` handles cases where the phase name contains special characters or spaces.
- The method `_extract_nodeid` correctly extracts node IDs for multiple phases in a single test.
- The method `_extract_nodeid` returns the correct node ID when the `include_phase` parameter is set to 'all'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	4 lines (ranges: 44-45, 216-217)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test should filter out setup phase when include_phase=run.

Why Needed: This test prevents a potential bug where the test may incorrectly extract node IDs for tests in the setup phase, leading to incorrect coverage analysis.

Key Assertions:

- The function _extract_nodeid() is called with the correct argument 'test.py::test_foo|setup'.
- The result of the call is not None (i.e., it does not return a node ID).
- The extracted node ID matches the expected pattern for tests in the setup phase.
- The test passes if the function correctly extracts node IDs without returning any 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/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the ability to extract nodeid from run phase context.

Why Needed: This test prevents a potential bug where the nodeid is not extracted correctly from the run phase context.

Key Assertions:

- The function _extract_nodeid() of CoverageMapper returns the correct nodeid for the given input.
- The nodeid returned by _extract_nodeid() matches the expected value.
- The nodeid is present in the run phase context.
- The nodeid does not contain any unnecessary characters or whitespace.
- The nodeid is correctly formatted as a string.
- The function _extract_nodeid() handles invalid input gracefully.

COVERAGE

src/pytest_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: 413-414, 417, 421-423)

📄 tests/test_coverage_map_maximal.py

9 tests

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

AI ASSESSMENT

Scenario: Test that the mapper extracts all contexts for a file with maximal coverage.

Why Needed: This test prevents regression where the mapper does not extract all contexts for files with maximal coverage.

Key Assertions:

- The function _extract_contexts should return all paths in mock_data.contexts_by_lineno for the given input.
- The function _extract_contexts should include 'test_app.py::test_one' and 'test_app.py::test_two' in the result.
- The function _extract_contexts should correctly count lines 1 and 2 for 'app.py'.
- The function _extract_contexts should return a list with one element containing 'test_app.py::test_one'.
- The function _extract_contexts should include all files from mock_data.measured_files in the result.
- The function _extract_contexts should correctly extract contexts for files with maximal 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/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: 413-414, 417, 421-423)
src/pytest_llm_report/util/fs.py	17 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64, 100, 103, 111-112, 116, 123)
src/pytest_llm_report/util/ranges.py	13 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65-67)

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

AI ASSESSMENT

Scenario: Test that the `extract_contexts` method handles data with no test contexts correctly.

Why Needed: This test prevents a regression where the coverage map is not generated for files without test contexts.

Key Assertions:

- The `_extract_contexts` method should return an empty dictionary when given mock_data with no measured files.
- The `contexts_by_lineno` method should return an empty dictionary for all files in mock_data.
- mock_data.measured_files.return_value should not contain any values.
- mock_data.contexts_by_lineno.return_value should be an empty dictionary.
- result should be equal to {} when given mock_data with no measured files.
- The `extract_contexts` method should handle data with no test contexts correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	14 lines (ranges: 44-45, 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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the `CoverageMapper` with different node id variants.

Why Needed: To ensure that the test coverage is accurate and comprehensive, especially when there are missing lines in the code.

Key Assertions:

- The `_extract_nodeid` method returns the expected node id for each line.
- The `NoneType` exception is raised when a line has no matching phase.
- The context without a pipe (`|`) is correctly identified as having no matching phase.
- The test coverage is accurate and comprehensive, including missing lines in the 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the function does not fail when no coverage files exist.

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

Key Assertions:

- The function should return None for _load_coverage_data() without any .coverage files.
- The number of warnings should be 1, with code 'W001'.
- The first warning should have the correct code 'W001'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	9 lines (ranges: 44-45, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test covers the scenario where a coverage file cannot be read successfully.

Why Needed: This test prevents a potential regression where the CoverageMapper does not handle errors reading coverage files correctly.

Key Assertions:

- The function _load_coverage_data() should return None when an error occurs while reading the coverage data.
- Any warnings generated by the mapper should contain the message 'Failed to read coverage data'.
- The mapper's warnings list should not be empty after this test is 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test should handle parallel coverage files from xdist and verify that it correctly updates the coverage data.

Why Needed: This test prevents regression in handling parallel coverage files from xdist, which could lead to incorrect coverage data being reported.

Key Assertions:

- The mock instances of `CoverageData` returned by `mock_data_cls.side_effect` should have been updated at least twice.
- The `update` method of `CoverageData` should have been called on the first instance.
- The `update` method of `CoverageData` should have been called on the second instance.
- The `update` method of `CoverageData` should not be called on the third instance.
- The mock instances of `CoverageData` should not have been updated again after the test has finished.

COVERAGE

src/pytest_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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function should return an empty dictionary when `_load_coverage_data` is called with a `None` value.
- The function should not raise any exceptions when `_load_coverage_data` returns None.
- The `map_coverage` method should be able to handle the case where no coverage data is available without crashing or throwing an exception.
- The test should verify that the returned dictionary has zero keys (i.e., no coverage information).
- The function should not return a dictionary with any specific structure when `_load_coverage_data` returns None.
- The `map_coverage` method should be able to handle different types of coverage data (e.g., JSON, CSV, etc.) without 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/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: 413-414, 417, 421-423)

PASSED `tests/test_coverage_map_maximal.py::TestCoverageMapperMaximal::test_map_source_coverage_analysis_error` 2ms 5

AI ASSESSMENT

Scenario: The test verifies that the CoverageMapperMaximal class handles analysis failures by skipping source coverage files with errors.

Why Needed: This test prevents a regression where the CoverageMapperMaximal class incorrectly includes error messages in its output.

Key Assertions:

- The mock_cov.analysis2 method is called and it should raise an exception.
- The mock_data.measured_files.return_value is set to ['app.py']
- The mock_cov.get_data.return_value is set to mock_data
- The entries list returned by mapper.map_source_coverage() should be empty.
- The length of the entries list should not exceed 0.
- An exception is raised in the analysis2 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	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: 413-414, 417, 421-423)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116, 123)

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

AI ASSESSMENT

Scenario: Test 'map_source_coverage_comprehensive' verifies that the coverage mapper correctly maps source code to coverage data.

Why Needed: This test prevents regression in coverage calculation when analyzing multiple files with different statements and coverage percentages.

Key Assertions:

- The function `map_source_coverage` should return one entry for each file in `mock_cov.get_data()` with the correct number of statements, covered percentage, missed percentage, and coverage percent.
- The entries returned by `map_source_coverage` should have a 'file_path' attribute set to the expected file path ('app.py')
- The entries returned by `map_source_coverage` should have a 'statements' attribute equal to 3 (the number of statements in 'app.py')
- The entries returned by `map_source_coverage` should have a 'covered' attribute equal to 2 (the coverage percentage for 'app.py')
- The entries returned by `map_source_coverage` should have a 'missed' attribute equal to 1 (the number of statements not covered in 'app.py')
- The entries returned by `map_source_coverage` should have a 'coverage_percent' attribute equal to 66.67% (the coverage percentage for 'app.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/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: 413-414, 417, 421-423)
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 creates a Warning with the correct code and message.

Why Needed: This test prevents a potential bug where the make_warning function returns an unknown warning without providing any additional information.

Key Assertions:

- The Warning object created by make_warning should have a 'code' attribute equal to WarningCode.W001_NO_COVERAGE.
- The Warning object created by make_warning should have a 'message' attribute that contains the string 'No .coverage file found'.
- The Warning object created by make_warning should have a 'detail' attribute that is 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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that warning codes have correct values.

Why Needed: This test prevents a potential regression where the warning code values are not correctly validated, potentially leading to incorrect handling of warnings in downstream code.

Key Assertions:

- {'message': "The value of WarningCode.W001_NO_COVERAGE is equal to 'W001'.", 'description': 'Expected value of WarningCode.W001_NO_COVERAGE'}
- {'message': "The value of WarningCode.W101_LLM_ENABLED is equal to 'W101'.", 'description': 'Expected value of WarningCode.W101_LLM_ENABLED'}
- {'message': "The value of WarningCode.W201_OUTPUT_PATH_INVALID is equal to 'W201'.", 'description': 'Expected value of WarningCode.W201_OUTPUT_PATH_INVALID'}
- {'message': "The value of WarningCode.W301_INVALID_CONFIG is equal to 'W301'.", 'description': 'Expected value of WarningCode.W301_INVALID_CONFIG'}
- {'message': "The value of WarningCode.W401_AGGREGATE_DIR_MISSING is equal to 'W401'.", 'description': 'Expected value of WarningCode.W401'}

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the `to_dict()` method of `Warning` class to ensure it correctly converts warnings into a dictionary.

Why Needed: This test prevents a potential bug where the warning information is not properly extracted from the `Warning` object and stored in the dictionary.

Key Assertions:

- The `code` attribute of the `Warning` object should be set to 'W001' when converting it to a dictionary.
- The `message` attribute of the `Warning` object should be set to 'No coverage' when converting it to a dictionary.
- The `detail` attribute of the `Warning` object should be set to 'some/path' when converting it to a dictionary.
- When setting `detail`, the test checks that its value is not empty.
- The `code` attribute of the resulting dictionary should match the expected value 'W001'.
- The `message` attribute of the resulting dictionary should match the expected value 'No coverage'.
- The `detail` attribute of the resulting dictionary should match the expected value 'some/path'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	6 lines (ranges: 70-72, 74-76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

📄 tests/test_errors_maximal.py

6 tests

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

AI ASSESSMENT

Scenario: Test verifies that a warning with the standard message is created when known code is used.

Why Needed: This test prevents a potential bug where warnings are not generated for known code, potentially causing unexpected behavior or errors.

Key Assertions:

- 'WarningCode.W101_LLM_ENABLED' should be equal to WarningCode.W101_LLM_ENABLED
- 'WARNING_MESSAGES[WarningCode.W101_LLM_ENABLED]' should be equal to WARNING_MESSAGES[W101_LLM_ENABLED]
- w.detail is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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 prevents a bug by ensuring the fallback message is used for unknown code.

Why Needed: This test prevents a regression where the fallback message is not used for unknown code, potentially leading to incorrect error messages in future versions of the library.

Key Assertions:

- The `make_warning` function should be called with the correct warning code (`WarningCode.W001_NO_COVERAGE`) when an unknown code is encountered.
- The expected fallback message for unknown code (`Unknown warning.`) should match the actual returned message.
- The `WARNING_MESSAGES` dictionary should contain the original message before it was modified by the test.
- After restoring the original message, the updated `WARNING_MESSAGES` dictionary should still contain the correct value.
- The `missing_code` variable should hold the valid warning code (`WarningCode.W001_NO_COVERAGE`) that is used when an unknown code is encountered.
- The `old_message` variable should be set to the expected fallback message (`Unknown warning.`) before calling `make_warning` with the invalid code.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test 'test_make_warning_with_detail' verifies creating a warning with detail.

Why Needed: This test prevents the creation of warnings without detail, which can lead to misleading error messages.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Ensures that enum values are correctly converted to strings.

Why Needed: Prevents a potential bug where enum values are not properly converted to strings.

Key Assertions:

- assert isinstance(code.value, str)
- assert code.value.startswith('W')
- code.value should be a string (not an integer or other type)
- code.value should start with 'W' (the warning code prefix)
- WarningCode is correctly defined and accessible within the test scope

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the warning to dictionary serialization without detail.

Why Needed: Prevents a potential bug where warnings are not properly serialized to dictionaries, potentially leading to incorrect data in logs or reports.

Key Assertions:

- The `to_dict()` method of the `Warning` class returns a dictionary with the correct keys and values.
- The keys 'code' and 'message' are present in the returned dictionary.
- The value for key 'code' is set to the expected string value 'W001'.
- The value for key 'message' is set to the expected string value '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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test warning to dictionary with detail should be performed.

Why Needed: This test prevents a potential bug where the Warning data class is not properly serialized into a dictionary, potentially causing issues when using it in certain scenarios.

Key Assertions:

- The 'to_dict()' method of the Warning class returns a dictionary with the correct keys.
- The 'code' key in the returned dictionary has the expected value.
- The 'message' key in the returned dictionary has the expected value.
- The 'detail' key in the returned dictionary has the expected value.
- The 'WarningCode.W001_NO_COVERAGE' attribute is correctly converted to a string.
- The 'WarningCode.W001_NO_COVERAGE' attribute is not 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/errors.py	6 lines (ranges: 70-72, 74-76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

📄 tests/test_fs.py

12 tests

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

AI ASSESSMENT

Scenario: Verifies that the function returns False for non-python file extensions.

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

Key Assertions:

- The function should return False when given a non-.py file extension (e.g. foo/bar.txt).
- The function should return False when given a non-.pyc file extension (e.g. foo/bar.pyc).
- The function should raise an AssertionError when given a non-python file.
- The function should not incorrectly identify .pyc files as non-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: 413-414, 417, 421-423)
src/pytest_llm_report/util/fs.py	1 lines (ranges: 79)

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

AI ASSESSMENT

Scenario: Verifies that the `is_python_file` function returns True for a `.py` file.

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

Key Assertions:

- The function should return `True` for files with extensions like `.py`.
- The function should ignore other file extensions (e.g., `.txt`, `.js`) and only check for `.py` files.
- The function should raise an error or handle the case where it's not a Python 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/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/fs.py	1 lines (ranges: 79)

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

AI ASSESSMENT

Scenario: Test makes absolute path relative by creating a subdirectory and making the specified file within it.

Why Needed: This test prevents a potential bug where the `make_relative` function fails to create the expected output when dealing with files in subdirectories.

Key Assertions:

- The `make_relative` function should be able to correctly make the specified file relative to the provided path.
- The `parent.mkdir(parents=True, exist_ok=True)` call should not raise an exception if the directory already exists.
- The `touch()` call should create a new file with the correct name and permissions.
- The `make_relative()` function should return the expected output string ('subdir/file.py') after making the specified path relative.
- The `assert` statement should fail when the actual output does not match the expected output.
- The test should be able to handle cases where the file is created in a subdirectory with an existing parent directory.
- The test should be able to handle cases where the file has incorrect permissions or ownership after being made relative.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 does not normalize the 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 handle cases where the input path has multiple levels of nesting correctly.
- The function `make_relative` should not modify the file system in any way, even if it returns a normalized path.
- The function `make_relative` should raise an error when given invalid input (e.g. non-string base or empty string).
- The function `make_relative` should correctly handle cases where the input path is absolute (i.e. starts with '/').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/fs.py	7 lines (ranges: 30, 33, 36, 39, 42, 55-56)

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

AI ASSESSMENT

Scenario: The test verifies that the `normalize_path` function correctly handles already-normalized paths.

Why Needed: This test prevents a potential bug where an already normalized path would be incorrectly normalized back to its original form.

Key Assertions:

- assert normalize_path('foo/bar') == 'foo/bar'
- assert normalize_path('/foo/bar') == '/foo/bar'
- assert normalize_path('//foo/bar') == '//foo/bar'
- assert normalize_path('///foo/bar') == '///foo/bar'
- assert normalize_path('foo//bar') == 'foo//bar'
- assert normalize_path('foo/./bar') == 'foo/./bar'
- assert normalize_path('/foo/./bar') == '/foo/./bar'
- assert normalize_path('//foo/./bar') == '//foo/./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/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

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

AI ASSESSMENT

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

Why Needed: This test prevents a bug where the function fails to convert backslashes to forward slashes, potentially leading to incorrect path comparisons or file system operations.

Key Assertions:

- assert normalize_path('foo\bar') == 'foo/bar'
- assert normalize_path('/foo/bar') == '/foo/bar'
- assert normalize_path('foo/\bar') == 'foo/bar'
- assert normalize_path('foo//bar') == 'foo/bar'
- assert normalize_path('foo/./bar') == 'foo/bar'
- assert normalize_path('foo/../bar') == 'foo/bar'
- assert normalize_path('/a/b/c') == '/a/b/c'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

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

AI ASSESSMENT

Scenario: tests/test_fs.py::TestNormalizePath::test_strips_trailing_slash

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

Key Assertions:

- assert normalize_path('foo/bar/') == 'foo/bar'
- normalize_path('foo/').should_return('/')
- normalize_path('a/b/c/').should_return('a/b/c')

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 a potential bug where paths matching custom patterns are incorrectly skipped.

Key Assertions:

- The function should return True for the 'tests/conftest.py' file and False for the 'src/module.py' file when exclude_patterns = ['test*']
- The function should correctly skip the 'tests/conftest.py' file due to the custom exclusion pattern
- The function should not incorrectly skip the 'src/module.py' file due to the custom exclusion pattern

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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

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: 413-414, 417, 421-423)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116, 123)

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

AI ASSESSMENT

Scenario: tests/test_fs.py::TestShouldSkipPath::test_skips_git verifies that the function should skip a .git directory.

Why Needed: This test prevents a potential issue where the function incorrectly identifies a .git directory as a file and thus skips it.

Key Assertions:

- assert should_skip_path('.git/objects/foo') is True
- should be equal to False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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

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: 413-414, 417, 421-423)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the `should_skip_path` function incorrectly identifies venv directories as paths to Python 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: 413-414, 417, 421-423)
src/pytest_llm_report/util/fs.py	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

📄 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 a potential bug where the rate limiter does not clear request and token usage records for past requests, leading to incorrect tracking of usage.

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 add a record for the past request.
- _token_usage.append((time.monotonic() - 61, 100)) should add a record with token usage for the past request.
- limiter._prune(time.monotonic()) should clear _request_times and _token_usage records for the past request.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/gemini.py	11 lines (ranges: 39-42, 81-85, 87-88)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the RPM limit configuration and its effect on the rate limiter.

Why Needed: This test prevents a potential bug where the rate limiter becomes unavailable after setting an incorrect `requests_per_minute` value.

Key Assertions:

- The rate limiter should be available again after calling `next_available_in(1)`.
- The rate limiter should not exceed 60 seconds when called with `next_available_in(1)`.
- The rate limiter's availability and time limit should match the expected values for 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/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents a regression when the token limit is exceeded.

Why Needed: This test prevents a bug where the rate limiter does not correctly handle cases when the token limit is exceeded.

Key Assertions:

- The next available time point should be greater than 0.
- The number of tokens used in the last 20 seconds should be less than or equal to 10.
- The number of tokens usage should increase by 1 after recording 10 new tokens.

COVERAGE

src/pytest_llm_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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the `wait_for_slot` method to ensure it sleeps as expected when a request is recorded.

Why Needed: This test prevents a potential issue where the rate limiter does not sleep before checking if there are available slots.

Key Assertions:

- The `wait_for_slot` method should call `time.sleep` with an argument equal to the number of requests per minute specified in the configuration.
- The `wait_for_slot` method should assert that `mock_sleep` was called exactly once during the test.
- The `wait_for_slot` method should not assert anything if no requests have been recorded yet (i.e., it does not sleep).
- The `wait_for_slot` method should only assert that `mock_sleep` was called when a request is recorded and there are available slots.
- The rate limiter's internal state should be updated correctly 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/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: 413-414, 417, 421-423)`

📄 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 does not attempt to record tokens when there are no tokens available.

Why Needed: This test prevents a potential regression where the rate limiter might incorrectly assume there are tokens available and attempt to record them, leading to incorrect usage statistics.

Key Assertions:

- The length of `_token_usage` is 0 after calling `record_tokens(0)`.
- The number of records in `_token_usage` remains unchanged even after the first call to `record_tokens(0)`.
- The rate limiter does not attempt to record tokens when there are no tokens available.
- The rate limiter correctly handles a zero token count without attempting to record additional tokens.
- The test passes with an empty `_token_usage` 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/gemini.py	6 lines (ranges: 39-42, 66-67)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that the test raises a RateLimitExceeded exception when exceeding the daily limit.

Why Needed: Prevents regression in case of excessive requests per day.

Key Assertions:

- The function `wait_for_slot` should raise `_GeminiRateLimitExceeded` with a message indicating that the requested number of slots exceeds the daily limit.
- The function `wait_for_slot` should not return immediately after raising the exception, allowing for further processing or retries.
- The function `record_request` should be called before attempting to wait for a slot.
- The function `wait_for_slot` should check if the requested number of slots exceeds the daily limit and raise an exception accordingly.
- The function `wait_for_slot` should not allow the test to complete without exhausting the rate limiter.
- The function `record_request` should be called before attempting to wait for a slot, allowing for any necessary processing or retries.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the TPM fallback wait time is correctly calculated when filling up the rate limiter.

Why Needed: This test prevents a potential regression where the TPM fallback wait time is too short, causing the rate limiter to fail or slow down unnecessarily.

Key Assertions:

- The value of `wait` should be greater than zero.
- The sum of `tokens_used` and `request_tokens` should exceed the rate limit.
- If `token_usage` is empty, then `wait` should still be greater than zero.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

PASSED tests/test_gemini_coverage_v2.py::test_gemini_provider_rpm_cooldown 598ms 6

AI ASSESSMENT

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

Why Needed: To prevent the test from passing when the RPM rate limit is exceeded on the first call.

Key Assertions:

- The 'models/gemini-pro' model should be present in the cooldowns dictionary.
- The value of the 'models/gemini-pro' model in the cooldowns dictionary should be greater than 1000.0 seconds.
- The time it takes to reach the cooldown threshold after a failed rate limit exceeded call should be less than or equal to 1 second.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

📄 tests/test_gemini_provider.py

5 tests

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

AI ASSESSMENT

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

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

Key Assertions:

- The annotation should contain the correct scenario 'Recovered Scenario'.
- The mock post call count should be equal to 2.
- The annotation should not have an error.
- The annotation's scenario attribute should match the expected value 'Recovered Scenario'.
- The annotation's status code should be 200 for the second successful response.
- The annotation should contain a valid model name 'models/m1'.

COVERAGE

src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that _annotate_internal returns the correct LlmAnnotation object when _call_gemini returns text and tokens.

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

Key Assertions:

- assert annotation.scenario == 'Success Scenario'
- assert not annotation.error
- assert mock_parse.return_value.scenario == 'Success Scenario'
- assert mock_parse.return_value.text == 'Success Scenario'
- assert mock_parse.return_value.tokens == []
- assert mock_get.return_value.json.return_value['models'][0]['name'] == 'models/gemini-1.5-flash'
- assert mock_get.return_value.json.return_value['rateLimits'][0]['value'] == 15
- assert mock_post.return_value.json.return_value['candidates'][0]['content']['parts'][0]['text'] == 'Success 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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Tests the availability of the Gemini provider when environment variables are set.

Why Needed: The test prevents a potential bug where the provider's availability is incorrectly reported as available when it should be unavailable due to an incorrect API token.

Key Assertions:

- provider._check_availability() should return False when environment variable GEMINI_API_TOKEN is not provided.
- provider._check_availability() should return True when environment variable GEMINI_API_TOKEN is provided with a valid API 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/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: 413-414, 417, 421-423)

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that the rate limiter does not block requests for a short period after initial requests.

Why Needed: This test prevents a potential issue where the rate limiter blocks subsequent requests immediately after the first two, potentially causing unexpected behavior or errors in downstream applications.

Key Assertions:

- The next_available_in method returns 0.0 before the third request is recorded.
- The next_available_in method returns a value between 0 and 60.0 for the third request.
- The wait time after the first two requests is less than or equal to 60.0 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: 413-414, 417, 421-423)

📄 tests/test_hashing.py

13 tests

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

AI ASSESSMENT

Scenario: Test that different configuration providers produce different hashes.

Why Needed: This test prevents a bug where the same configuration provider is used with different inputs, resulting in identical hashes.

Key Assertions:

- config1 and config2 should have different hashes
- The hash of config1 should not be equal to the hash of config2
- The hash of config1 should not be equal to a hash generated by another configuration provider (ollama)
- The hash of config2 should not be equal to a hash generated by another configuration provider (none)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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: This test prevents a potential issue where the hash might be too long and cause performance issues or security vulnerabilities.

Key Assertions:

- The length of the computed hash should be exactly 16 characters.
- The computed hash should not exceed 15 characters due to padding.
- No leading zeros are present in the computed hash.
- All non-zero digits are present in the computed hash.
- The first character is a zero (0).
- The second character is also a zero (0).
- All other characters are non-zero and distinct.
- There are no duplicate characters in the computed hash.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 96-101, 103-104)

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

AI ASSESSMENT

Scenario: Test that the computed SHA-256 hash of a file matches its content hash when the same file is used for both computations.

Why Needed: This test prevents a bug where the computed file hash does not match the content hash due to differences in file system or storage implementation.

Key Assertions:

- The computed SHA-256 hash of the file should be equal to its content hash.
- The content hash of the file should be equal to the computed SHA-256 hash.
- The computed file hash should match the content hash when the same file is used for both computations.
- The computed SHA-256 hash of a file with different content than its content hash should not match the content hash.
- The computed file hash should be consistent across multiple runs of the test.
- The content hash of a file with different content than its content hash should also be consistent across multiple runs 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/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/hashing.py	6 lines (ranges: 32, 44-48)

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

AI ASSESSMENT

Scenario: Verify that the test hashes a file correctly.

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

Key Assertions:

- The length of the computed SHA-256 hash should be exactly 64 bytes.
- The hash value should match the expected output provided by `compute_file_sha256(path)`.
- Any errors or exceptions raised during the computation process should not affect the overall test 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: 413-414, 417, 421-423)
src/pytest_llm_report/util/hashing.py	5 lines (ranges: 44-48)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the same key produces the same signature for different inputs.

Key Assertions:

- The computed HMAC signature should be different for two different keys.
- The computed HMAC signature should not be equal to the expected signature for key1.
- The computed HMAC signature should not be equal to the expected signature for key2.
- The computed HMAC signature should have a different value when compared to the expected signature for key1.
- The computed HMAC signature should have a different value when compared to the expected signature for key2.
- The computed HMAC signature should be different from the expected signature for both keys.
- The computed HMAC signature should not match the expected signature for key1 and key2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 61)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential issue where an attacker could manipulate the signature by tampering with the input data or the secret key.

Key Assertions:

- The length of the computed HMAC should be 64 bytes.
- The computed HMAC should not be empty.
- The computed HMAC should contain all characters from the input data.
- The computed HMAC should contain the correct padding.
- The computed HMAC should include the secret key.
- The computed HMAC should have a length of 64 bytes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 61)

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

AI ASSESSMENT

Scenario: Test that the computed SHA-256 hashes are consistent for the same input.

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

Key Assertions:

- The two computed hashes should be equal.
- The hash values should have the same length (32 bytes).
- The first character of each hash should match.
- The second character of both hashes should match.
- All characters in both hashes except for the last one should match.
- The last character of both hashes should match.
- The hash values should be identical when compared using a hashing algorithm like SHA-256.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 32)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the hash length may be less than 64 characters, potentially causing issues with certain applications or libraries that require this length.

Key Assertions:

- The length of the computed hash should be exactly 64 hexadecimal characters.
- The computed hash should not have any leading zeros.
- All non-zero hexadecimal digits in the hash should appear consecutively.
- No whitespace or punctuation should be present in the hash.
- The hash should not contain any duplicate bytes.
- All hexadecimal digits should be either 0-9, A-F, a-f, or +/.
- The hash should start with an even number of hexadecimal digits for SHA-256.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 32)

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

AI ASSESSMENT

Scenario: Verifies that the `get_dependency_snapshot` function includes the 'pytest' package as part of its output.

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' section of the snapshot.
- The 'pytest' package should be listed as an item in the 'dependencies' section of the snapshot.
- The 'pytest' package should be included in the output of the `get_dependency_snapshot` function.
- The presence of 'pytest' in the dependency snapshot is required for the test to pass.
- Including 'pytest' in the dependency snapshot ensures accurate testing and reproducibility of the code.
- Without 'pytest', the test would fail due to missing dependencies.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`
`src/pytest_llm_report/util/hashing.py`	`8 lines (ranges: 113-114, 116-121)`

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the function might return an incorrect data type or format.

Key Assertions:

- snapshot is of type dict
- snapshot has correct keys (e.g. 'dependencies', 'versions')
- snapshot does not contain any missing keys
- snapshot contains only valid package information
- snapshot is a dictionary with the expected structure
- snapshot can be converted to a dictionary using `json.dumps()`
- snapshot is not empty or None

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`
`src/pytest_llm_report/util/hashing.py`	`8 lines (ranges: 113-114, 116-121)`

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

AI ASSESSMENT

Scenario: Test 'test_loads_key' verifies that the HMAC key can be loaded from a file.

Why Needed: This test prevents a bug where the HMAC key is not properly loaded from a file due to incorrect file path or missing file.

Key Assertions:

- The `hmac_key_file` attribute of the `Config` object should point to the expected file location.
- The `load_hmac_key` function should be able to successfully load the HMAC key from the specified file.
- The loaded HMAC key should match the expected value 'my-secret-key'.
- The configuration object `config` should have a `hmac_key_file` attribute pointing to the correct file location.
- An error message indicating that the file was not found or could not be read should not be raised when loading the HMAC key.
- The loaded HMAC key should contain only the expected bytes 'my-secret-key'.
- The configuration object `config` should have a `hmac_key_file` attribute pointing to the correct file location and the HMAC key should match it.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/hashing.py	5 lines (ranges: 73, 76-77, 80-81)

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

AI ASSESSMENT

Scenario: Test that a missing key file returns None when provided.

Why Needed: This test prevents a potential bug where the HMAC key is not loaded due to an empty or non-existent key file.

Key Assertions:

- The function `load_hmac_key` should return `None` if the `hmac_key_file` parameter is set to `nonexistent.key`.
- The test should fail when a missing key file is provided.
- The expected output of `load_hmac_key` should be `None` in this case.
- The function call with an empty string as a key file path should raise a `ValueError`.
- A `KeyError` exception should be raised when trying to load the HMAC key from a non-existent file.
- The test should not fail if the `hmac_key_file` parameter is set to a valid file path but the key is missing.
- The function call with a valid key file path and an empty string as a key file should return the loaded 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: 413-414, 417, 421-423)
src/pytest_llm_report/util/hashing.py	4 lines (ranges: 73, 76-78)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the function does not handle the case of an empty or missing key file configuration.

Key Assertions:

- The `config` object passed to `load_hmac_key` is not None.
- The `key` variable is None after calling `load_hmac_key(config).
- No exception is raised when no key file is specified in the configuration.
- The function does not throw an error or raise a meaningful exception when given an empty or missing key file configuration.
- The `load_hmac_key` function's behavior changes unexpectedly without providing clear documentation or warnings.
- The test does not verify that the function throws a `ConfigError` with a specific message when no key file is specified.
- The test does not cover all possible edge cases, such as a missing key file in a configuration object with other settings.
- The test does not provide any information about what happens to the configuration object or the loaded HMAC key after calling `load_hmac_key(config).'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 the default aggregation configuration.

Why Needed: Prevents a regression where aggregation defaults to an empty directory and does not include historical data.

Key Assertions:

- config.aggregate_dir should be None.
- config.aggregate_policy should be 'latest'.
- config.aggregate_include_history should be False.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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: Prevents a regression where the default capture failed output was accidentally set to True.

Key Assertions:

- The `capture_failed_output` field in the test configuration is set to `False`.
- The value of `capture_failed_output` in the test configuration matches the expected default value.
- The captured output does not contain any failed messages when `capture_failed_output` is False.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Tests the default context mode for minimal configuration.

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

Key Assertions:

- The function get_default_config() returns an LLMContextMode object with value 'minimal'.
- The llm_context_mode attribute of config.llm is equal to 'minimal'.
- config.llm_context_mode is a string and its value is 'minimal'.
- get_default_config().llm_context_mode is not None.
- config.llm_context_mode is an object with contextMode property set to minimal.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: Prevent regression where LLM is enabled by default due to a bug or change in the configuration.

Key Assertions:

- The `is_llm_enabled()` method returns False.
- The `get_default_config()` function returns an instance with `is_llm_enabled()` set to True.
- The `config` object has a non-zero value for `llm_enabled` attribute.
- The `config` object does not have any attributes that could cause it to be enabled by default.
- The `get_default_config()` function returns an instance with a different configuration than expected.
- The `is_llm_enabled()` method is called on the `config` object without checking its current state first.
- The `is_llm_enabled()` method checks for the presence of certain attributes or values in the `config` 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`	`4 lines (ranges: 123, 163, 252, 261)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

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

AI ASSESSMENT

Scenario: The `TestConfigDefaults` class's `test_omit_tests_default_true` method verifies that the `omit_tests_from_coverage` attribute of the configuration object returned by `get_default_config()` is set to `True`.

Why Needed: This test prevents a potential issue where tests are not being omitted from coverage analysis due to an incorrect or missing configuration setting.

Key Assertions:

- The `omit_tests_from_coverage` attribute of the configuration object returned by `get_default_config()` is set to `True`.
- The value of `omit_tests_from_coverage` is indeed `True` for the given test case.
- A test is not being omitted from coverage analysis due to an incorrect or missing configuration setting.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`3 lines (ranges: 123, 163, 261)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the provider setting is not correctly handled when it defaults to 'none'.

Key Assertions:

- config.provider should be None.
- assert config.provider == 'none' after calling get_default_config().

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

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

Key Assertions:

- The 'secret' keyword is present in any glob pattern used to exclude files.
- The '.env' file is also excluded from the list of patterns.
- Any other secret files that may have been added by external tools are not included in the exclusion 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	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_integration_gate.py::TestFullPipeline::test_deterministic_output` 7ms 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/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: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	117 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where the full pipeline is not producing any reports.

Key Assertions:

- The total count of tests in the report should be zero.
- The summary section of the report should have a 'total' key with a value of zero.
- The data from the report.json file should contain a 'summary' section with a 'total' key equal to zero.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	67 lines (ranges: 235-237, 239, 241, 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516-518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	118 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_integration_gate.py::TestFullPipeline::test_html_report_generation` 35ms 6

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the HTML report is missing or incorrect due to a bug in the ReportWriter class.

Key Assertions:

- The HTML file exists at the specified path.
- The HTML file contains the expected content with the test result.
- The test result string 'test_pass' is present in the HTML file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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` 59ms 7

AI ASSESSMENT

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

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

Key Assertions:

- The report writer is configured with a path to a 'report.json' file.
- The report is written successfully and exists at the specified path.
- The schema version of the report matches the expected value.
- The summary statistics (total, passed, failed, skipped) are correctly reported.
- The 'tests/test_a.py::test_one', 'tests/test_a.py::test_two', and 'tests/test_b.py::test_skip' nodes have been successfully executed with their respective outcomes.
- The test has not been skipped due to a timeout or other error.

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: 413-414, 417, 421-423)
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` class has a report root with required fields.

Why Needed: This test prevents a regression where the schema version, run meta, summary, and tests are missing from the report root.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that `RunMeta` has aggregation fields when it's not aggregated.

Why Needed: This test prevents regression where `RunMeta` does not have aggregation fields even if it's not aggregated.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that RunMeta has status fields.

Why Needed: This test prevents a potential bug where the run metadata does not contain all necessary status fields.

Key Assertions:

- status_fields_exist
- exit_code_field_exists
- interrupted_field_exists
- collect_only_field_exists
- collected_count_field_exists
- selected_count_field_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	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: 413-414, 417, 421-423)

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 should be defined.
- The schema version should contain at least one dot (.) character.
- The schema version should be in a valid 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: 413-414, 417, 421-423)`

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

AI ASSESSMENT

Scenario: Test verifies that `TestCaseResult` has required fields.

Why Needed: This test prevents a potential bug where the `TestCaseResult` object is missing some required fields, potentially leading to incorrect analysis or reporting.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: 413-414, 417, 421-423)

📄 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 configuration is set to 'gemini'.

Why Needed: This test prevents a potential bug where the `get_provider` function does not return a valid provider.

Key Assertions:

- provider.__class__ == 'GeminiProvider'
- provider.model == 'gemini-1.5-flash'
- provider.name == 'GeminiProvider'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	10 lines (ranges: 52-53, 245, 247, 249, 252, 257, 262-263, 265)
src/pytest_llm_report/llm/gemini.py	7 lines (ranges: 134, 136-139, 141-142)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that the `get_provider` function returns an instance of LiteLLMProvider when a specific provider is specified.

Why Needed: This test prevents a potential bug where the correct provider is not returned due to incorrect configuration or missing dependencies.

Key Assertions:

- The `provider` attribute of the returned `LiteLLMProvider` instance should be set to 'litellm'.
- The `__class__.__name__` attribute of the returned `LiteLLMProvider` instance should match '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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: test_get_provider_with_none_provider returns NoopProvider.

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

Key Assertions:

- The function get_provider() should be able to create a NoopProvider instance when the 'provider' parameter is set to 'none'.
- The returned value of get_provider(config) should be an instance of NoopProvider.
- The assertion isinstance(provider, NoopProvider) should pass for the correct NoopProvider 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that OllamaProvider is returned when 'provider='ollama' in the configuration.

Why Needed: This test prevents a potential bug where the correct provider type (OllamaProvider) is not detected when using the 'provider=ollama' configuration.

Key Assertions:

- The function get_provider() returns an instance of OllamaProvider.
- The class name of the returned provider matches 'OllamaProvider'.
- The method __class__ checks if the returned provider is indeed an instance of OllamaProvider.
- The type of the provider is correctly identified as OllamaProvider.
- The configuration object passed to get_provider() has a valid provider value ('provider=ollama').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	8 lines (ranges: 52-53, 245, 247, 249, 252-253, 255)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test 'test_unknown_raises' verifies that an unknown provider raises a ValueError.

Why Needed: This test prevents the regression where an unknown provider is used without raising a ValueError.

Key Assertions:

- The function `get_provider(config)` should be called with a valid provider.
- The function `Config(provider='unknown')` should be created with an invalid provider.
- A ValueError should be raised when trying to get a provider from the unknown config.
- The error message 'unknown' should contain the string 'unknown' in lowercase.
- An AssertionError should be raised if the test is run without raising a ValueError.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that NoopProvider implements LlmProvider interface.

Why Needed: Prevents a bug where NoopProvider does not implement required methods of LlmProvider.

Key Assertions:

- should have required methods
- should be able to annotate
- should be able to check if available
- should have model name attribute
- should have config attribute

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the NoopProvider does not return any annotation for certain scenarios.

Key Assertions:

- annotation is of type LlmAnnotation
- annotation scenario is an empty string
- annotation why_needed is an empty string
- annotation 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: 413-414, 417, 421-423)

PASSED `tests/test_llm.py::TestNoopProvider::test_get_model_name_empty` 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/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that the NoopProvider instance is always available.

Why Needed: To ensure the provider's availability, which is crucial for its functionality.

Key Assertions:

- The `is_available()` method should return `True`.
- The `is_available()` method should not raise an exception.
- The `is_available()` method should be able to be called without raising any errors.
- The provider's availability can be checked using the `is_available()` method.
- The provider's availability is a critical aspect of its functionality and should not be compromised.
- The test should fail if the provider is not available, indicating a bug or regression.

COVERAGE

src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

📄 tests/test_llm_annotator.py

6 tests

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

AI ASSESSMENT

Scenario: The test verifies that the annotation summary is printed when annotations run.

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

Key Assertions:

- The function `test_annotate_tests_emits_summary` should print 'Annotated 1 test(s) via litellm' in the captured output.
- The provider `FakeProvider(LlmAnnotation)` should be used instead of the actual LLM annotation provider.
- The `get_provider` method of the `llm.annotator` module should return a `FakeProvider` instance when called with a configuration object.

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: 413-414, 417, 421-423)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

Scenario: Test that the progress of LLM annotations is reported correctly.

Why Needed: This test prevents regression where the progress of LLM annotations is not reported.

Key Assertions:

- The correct message should be printed when starting LLM annotations for a test.
- The correct message should include the name of the test that started the annotation.
- Each annotation should have its own unique message.
- The message should indicate the number of tests being annotated.
- The message should contain the provider used to annotate the test.
- The message should be printed before starting any annotations.
- Any error messages from annotations should not be printed.

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: 413-414, 417, 421-423)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

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

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

Key Assertions:

- The 'tests/test_a.py::test_a' node should be called with the provider.
- The 'tests/test_b.py::test_b' node should have an LLM annotation set to None.
- The 'tests/test_c.py::test_c' node should not have an LLM annotation set to None.
- The number of LLM annotations for all tests should be 1.
- The provider function should call the fake provider with the correct configuration.
- The calls made by the fake provider should only include the 'tests/test_a.py::test_a' node.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	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: 413-414, 417, 421-423)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the annotator exceeds the allowed number of requests per minute, potentially causing performance issues or errors.

Key Assertions:

- The provider's calls to `llm.annotator.get_provider` should be ['tests/test_a.py::test_a', 'tests/test_b.py::test_b']
- The sleep_calls list should contain [2.0] as the expected value

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: 413-414, 417, 421-423)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

Scenario: Test that annotation skips unavailable providers with a clear message.

Why Needed: To prevent skipping annotation of tests when an unavailable provider is detected.

Key Assertions:

- The 'is_available' method returns False for the unavailable provider.
- The annotation process should not proceed without checking the provider's availability.
- A clear error message should be displayed indicating that the provider is unavailable.
- The test should fail when an unavailable provider is encountered during annotation.
- The test should provide a descriptive error message explaining why the annotation skipped.
- The test should include the provider name in the failure message to identify the issue.

COVERAGE

src/pytest_llm_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: 413-414, 417, 421-423)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/cache.py	30 lines (ranges: 39-41, 53, 55-56, 58, 60-62, 68-73, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-81, 87-92, 97-98, 100, 102, 104, 115-122, 127, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 a regression where the schema is not enforced, potentially allowing invalid data to pass through.

Key Assertions:

- assert 'scenario' in required
- assert 'why_needed' in required

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

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

AI ASSESSMENT

Scenario: Test that AnnotationSchema.from_dict() correctly parses a dictionary with required keys.

Why Needed: Prevents dict-based schema creation from bypassing authentication mechanisms.

Key Assertions:

- checks password
- checks username

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/schemas.py`	`5 lines (ranges: 77-81)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

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

AI ASSESSMENT

Scenario: The test verifies that the AnnotationSchema class correctly handles an empty input.

Why Needed: This test prevents a potential bug where the AnnotationSchema class may throw an error or produce incorrect results when given an empty dictionary as input.

Key Assertions:

- schema.scenario = "" (empty string)
- schema.why_needed = "" (empty string)"" (empty string)
- schema.scenario in """ (empty string, list of strings)

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that the `ANNOTATION_JSON_SCHEMA` contains required fields.

Why Needed: This test prevents a potential bug where the schema is missing essential fields, potentially leading to errors or inconsistencies.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

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

AI ASSESSMENT

Scenario: Test AnnotationSchema::test_schema_to_dict verifies that the annotation schema is correctly serialized to a dictionary.

Why Needed: This test prevents regression by ensuring that the annotation schema can be properly converted into a JSON-like format.

Key Assertions:

- assertion 1: The 'scenario' key in the data dictionary should match the provided scenario string.
- assertion 2: The 'why_needed' key in the data dictionary should match the provided why_needed string.
- assertion 3: The 'key_assertions' key in the data dictionary is present and contains the expected list of assertions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 90-92, 94-96, 98)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The factory function should return a NoopProvider instance when the 'provider' parameter is set to 'none'.

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

Key Assertions:

- The config object passed to get_provider() has the correct value for the 'provider' key.
- The provider instance returned by get_provider() is an instance of NoopProvider.
- The assert isinstance() call passes with a NoopProvider instance as expected.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 245, 247, 249-250)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that the `NoopProvider` class correctly inherits from `LlmProvider`.

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

Key Assertions:

- The `provider` variable should be an instance of `LlmProvider`.
- The `provider` variable should not have any additional attributes or methods outside of the `LlmProvider` interface.
- The `provider` variable should not inherit from `Config` directly, but rather through its intended class hierarchy.
- The `NoopProvider` class should be correctly implemented to only provide a no-op service without any additional functionality.
- Any assertions made within the test should focus on verifying that the `LlmProvider` interface is respected by the `NoopProvider` implementation.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The NoopProvider should return an empty annotation when no tests are provided.

Why Needed: This test prevents a regression where the NoopProvider returns an empty annotation for no tests.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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 an LlmAnnotation-like object with the required key assertions.

Why Needed: This test prevents a potential regression where the annotate method does not return the expected annotations, potentially causing issues downstream in the testing pipeline.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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 non-empty result.

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

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the provider's handling of None context when annotating a TestCaseResult.

Why Needed: This test prevents potential bugs where the provider might fail or produce incorrect results when given a None context for an annotation.

Key Assertions:

- The result is not None after calling `provider.annotate(test, 'code', None)`.
- The provider can handle None context without raising an exception or producing unexpected results.
- The annotate method does not raise an error when given a None value for the 'code' field.
- The provider's behavior changes when given a None context for the annotation 'code'.
- The test case passes even if the provider encounters None context during execution.
- The result is not None after calling `provider.annotate(test, 'code', None)` with a valid value.
- The annotate method does not raise an error when given a non-None value for the 'code' field.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 413-414, 417, 421-423)

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 regression in the contract where providers are not annotated with a method.

Key Assertions:

- The provider has an attribute named 'annotate'.
- The provider is callable.
- The provider has a method named 'annotate'.
- The annotate method is defined on the provider object.
- The annotate method is correctly called when invoked.
- The annotate method is not None.
- The annotate method does not throw any exceptions.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

📄 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 is too large.

Why Needed: This test prevents a potential memory leak or performance issue caused by annotating contexts larger than expected.

Key Assertions:

- The `annotate` method should not be called with a context that exceeds the maximum allowed size.
- The `annotate` method should raise an exception when called with an invalid context.
- The `annotate` method should log a warning or error message when called with an invalid context.
- The `annotate` method should update the annotation metadata correctly even if the context is too large.
- The `annotate` method should not modify the original context but create a new one instead.
- A test case should be added to verify this scenario and prevent it from occurring in the future.

COVERAGE

src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The LiteLLMProvider annotates the missing dependency correctly.

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

Key Assertions:

- assert annotation.error == 'litellm not installed. Install with: pip install litellm'
- provider.annotate(test, 'def test_case(): assert True')
- test_case() should raise an exception when the provider cannot annotate it 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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that a missing API token prevents the annotation of a case with an annotated function.

Why Needed: This test verifies that setting the GEMINI_API_TOKEN environment variable before running tests will prevent the annotation of cases with annotated functions when the provider is not configured to use it.

Key Assertions:

- The `GEMINI_API_TOKEN` environment variable should be set before running the test.
- The `GEMINI_API_TOKEN` environment variable should be present in the configuration object.
- The annotation function should raise an error when the `GEMINI_API_TOKEN` is not set.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/gemini.py	12 lines (ranges: 134, 136-139, 141-142, 160-161, 167-169)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that tokens are recorded correctly by the Gemini provider.

Why Needed: Prevents regressions and ensures accurate token usage tracking.

Key Assertions:

- The 'status ok' assertion is included in the response metadata.
- The totalTokenCount key contains the correct value of 123.
- The candidates list includes a single record with text containing the expected JSON data.
- The usageMetadata key contains the correct rate limits information.
- The limiter object is not None, indicating that tokens were recorded correctly.
- The limiter._token_usage list has one element with the correct token count and value.
- The limiter._token_usage[0][1] equals 123, which 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/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that the LLM provider annotates retries on rate limits correctly.

Why Needed: This test prevents a potential regression where the LLM provider does not handle rate limit retrials correctly.

Key Assertions:

- The LLM provider should retry the annotation process after a rate limit is exceeded.
- The LLM provider should re-annotate the model with the updated parameters after a rate limit is exceeded.
- The LLM provider should update the model's annotations to reflect the new parameters after a rate limit is exceeded.
- The LLM provider should not retry the annotation process if the rate limit is not exceeded.
- The LLM provider should respect the rate limits set by the client and not retry the annotation process.
- The LLM provider should update the model's annotations with the correct parameters after a rate limit is exceeded, even if it's not explicitly retried.

COVERAGE

`src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The `annotate` method of the `GeminiProvider` class rotates models on a daily limit during testing.

Why Needed: This test prevents regression in the LLM model rotation feature by ensuring that models are rotated only once per day.

Key Assertions:

- The `rotate_models_on_daily_limit` method is called with an empty dictionary as its argument.
- A new model is created and added to the list of models.
- The `rotate_models_on_daily_limit` method is called again with a non-empty dictionary as its argument, which should not rotate any existing models.
- No exception is raised when rotating models on a daily limit.
- The number of models in the list remains unchanged after rotation on a daily limit.
- Models are rotated only once per day, regardless of the test duration.
- The `rotate_models_on_daily_limit` method is called with an empty dictionary as its argument during the first test run.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

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 annotations when the daily limit is exceeded.

Why Needed: This test prevents a regression where the `annotate` method fails to skip annotations due to an incorrect implementation of the daily limit check.

Key Assertions:

- The `annotate` method should not be called with more than one annotation per day.
- The `annotate` method should skip any annotation that is already being processed or has been skipped in the past day.
- The `annotate` method should raise an error if it tries to annotate a new annotation when the daily limit is exceeded.
- The `annotate` method should not be called with annotations that are not valid (e.g. invalid types, missing required arguments).
- Any annotations that have been skipped in the past day should still be skipped by the `annotate` method even if they are re-added.
- The `annotate` method should correctly handle cases where multiple annotations need to be annotated at once (e.g. multiple entities being processed)
- The `annotate` method should not silently ignore or crash when it encounters an annotation that is already being processed or has been skipped in the past day.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the annotate method correctly annotates a successful response with mock data.

Why Needed: Prevents regressions caused by missing or incorrect annotations in responses from LiteLLM providers.

Key Assertions:

- The annotation contains the correct scenario and why-needed information.
- The annotation includes the expected key assertions.
- The annotation has a non-zero confidence level.
- The captured model is correctly set for the test case.
- The system role of the message is 'system' as expected.
- The test login function is present in the response messages.
- The def test_login() function is present in the response messages.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 413-414, 417, 421-423)

PASSED `tests/test_llm_providers.py::TestGeminiProvider::test_exhausted_model_recovers_after_24h` 1ms 6

AI ASSESSMENT

Scenario: The test verifies that the exhausted model recovers after 24 hours.

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

Key Assertions:

- The recovered model should have the same accuracy as before exhaustion.
- The recovered model's inference time should be less than or equal to 24 seconds.
- The recovered model's memory usage should decrease by at least 50% compared to before exhaustion.
- The recovered model's computation time should decrease by at least 30% compared to before exhaustion.
- The recovered model's latency should decrease by at least 20% compared to before exhaustion.
- The recovered model's memory allocation should be reduced by at least 75% compared to before exhaustion.
- The recovered model's garbage collection should complete within the first 24 seconds of inference.

COVERAGE

src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The `fetch_available_models` method of the `GeminiProvider` class raises an error when there are no available models.

Why Needed: This test prevents a potential regression where the `fetch_available_models` method returns incorrect results or raises an exception due to an incomplete set of available models.

Key Assertions:

- The `fetch_available_models` method should return an empty list when there are no available models.
- The `fetch_available_models` method should raise a `GeminiProviderError` with a suitable error message when there are no available models.
- The `fetch_available_models` method should not return any results when there are no available models and the `GEMINIMODELS` environment variable is set to `None` or an empty string.
- The `fetch_available_models` method should raise a `GeminiProviderError` with a suitable error message when the `GEMINIMODELS` environment variable is not set or is an invalid value.
- The `fetch_available_models` method should return a list of available models with a length greater than 0 when there are available models and the `GEMINIMODELS` environment variable is set to a valid value.
- The `fetch_available_models` method should raise a `GeminiProviderError` with a suitable error message when the `GEMINIMODELS` environment variable is not set or is an invalid value, but there are available models.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/gemini.py	65 lines (ranges: 134, 136-139, 141-142, 286, 288-289, 292-296, 298-301, 303-304, 306-307, 352, 354-356, 358-361, 366-369, 380-383, 391, 393, 397-398, 402-408, 411, 414-416, 418-420, 423-424, 434, 436-438, 441-442)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression that may occur when the interval between model updates is changed.

Key Assertions:

- The `refresh_interval` attribute of the `GeminiProvider` instance is set to the expected value before each test call.
- After refreshing the model list, all models are present and up-to-date within the specified time frame.
- The `refresh_interval` attribute is updated correctly after each test call to reflect the new interval.
- No exceptions are raised when refreshing the model list with an invalid or outdated interval value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	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: 413-414, 417, 421-423)

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 handle the case where the user's token is refreshed, leading to a 401 error.

Key Assertions:

- Verify that the provider correctly sets the `litellm_token_refresh_command` attribute.
- Verify that the provider correctly sets the `litellm_token_refresh_interval` attribute.
- Verify that the provider attempts to refresh the token after the first failed attempt.
- Verify that the second successful attempt does not result in a 401 error.
- Verify that the correct tokens are captured and stored for future reference.
- Verify that the `call_count` variable is incremented correctly for each call to `fake_completion`.
- Verify that the `token_count` variable is incremented correctly for each call to `fake_run`.
- Verify that the `config` object contains the correct settings for the 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	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: 413-414, 417, 421-423)

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 in annotations.

Key Assertions:

- The annotation should contain an error message indicating a completion error.
- The error message should be 'boom'.
- The annotation should include the line number and function name of the completion error.
- The annotation should include the string 'boom' as the error message.
- The annotation should not ignore the completion error when it occurs in the test case.
- The annotation should surface the completion error in the output of the LiteLLMProvider.
- The annotation should indicate that the completion error is a result of the test case being executed.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

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: This test prevents the provider from silently failing when receiving an invalid key_assertions payload, allowing for better debugging and error messages.

Key Assertions:

- The 'response_data' parameter must be a dictionary.
- The 'json.dumps(response_data)' function should raise a TypeError if it cannot serialize the data.
- The 'response_data' dictionary should contain a 'key_assertions' key with a list of strings.
- The 'litellm' module should have a 'litellm' attribute that is a callable.
- The 'fake_litellm_response' function should return an instance of the 'FakeLiteLLMResponse' class.
- The 'json.dumps(response_data)' function should raise a TypeError if it cannot serialize the data.
- The 'response_data' dictionary should contain a 'key_assertions' key with a list of strings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

PASSED `tests/test_llm_providers.py::TestLiteLLMProvider::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 message is set to 'litellm not installed. Install with: pip install litellm'.
- The annotation indicates that the dependency was not found.
- The annotation provides a clear and concise error message.
- The test verifies that the provider reports an error for missing dependencies correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the annotate method returns an LlmAnnotation object with the correct scenario, why_needed, and key_assertions.

Why Needed: Prevents a regression where LiteLLMProvider is not correctly annotating successful responses from mock responses.

Key Assertions:

- The annotation has the correct scenario 'Checks login'.
- The annotation has the correct why_needed 'Stops regressions'.
- The annotation has the correct key_assertions ['status ok', 'redirect'].

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test the LiteLLM provider's API base passthrough functionality.

Why Needed: This test prevents regression in case the API base is not passed through correctly.

Key Assertions:

- Verify that the `api_base` attribute of the `LiteLLMProvider` instance is set to 'https://proxy.corp.com/v1'.
- Check if the `litellm_api_base` configuration option is set to a valid URL.
- Verify that the `FakeLiteLLMResponse` object returned by `fake_completion` has an `api_base` attribute equal to 'https://proxy.corp.com/v1'.

COVERAGE

src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test: tests/test_llm_providers.py::TestLiteLLMProvider::test_api_key_passthrough

Why Needed: This test prevents a bug where the API key is not passed through to the completion call of LiteLLM provider.

Key Assertions:

- The API key should be set to `TEST_KEY` when using the 'litellm' provider.
- The API key should be retrieved from environment variable `TEST_KEY` if it's not provided in the config.
- The API key should be passed through to the completion call of LiteLLM 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the LiteLLM provider detects installed module.

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

Key Assertions:

- The `is_available()` method of the `LiteLLMProvider` class should return `True` when the `litellm` module is available in the system's modules.
- When the `litellm` module is installed, the provider should be able to detect it and set its `is_available()` attribute accordingly.
- The provider should not raise any exceptions or errors if the `litellm` module is already loaded and available.
- If the `litellm` module is not installed, the provider should still be able to detect it correctly.
- The provider's behavior should change when the `litellm` module is removed from the system's modules.
- When the `litellm` module is imported dynamically using `import litellm`, the provider should also detect its presence.
- If the `litellm` module is not available in the system's modules, the provider should still be able to set its `is_available()` attribute correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 107, 110-111)
src/pytest_llm_report/llm/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: 413-414, 417, 421-423)

PASSED `tests/test_llm_providers.py::TestLiteLLMProvider::test_token_refresh_integration` 1ms 7

AI ASSESSMENT

Scenario: The test verifies that the LiteLLM provider uses TokenRefresher to refresh tokens in a dynamic token scenario.

Why Needed: This test prevents regression when the provider fails to refresh tokens due to insufficient or expired tokens.

Key Assertions:

- The `litellm_token_refresh_command` is set to 'get-token'.
- The `litellm_token_refresh_interval` is set to 3600 seconds.
- The `api_key` is captured and verified as 'dynamic-token-789' after the test case execution.
- A successful token refresh is expected within 1 hour (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/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verifies that the annotate fallbacks on context length error are handled correctly.

Why Needed: Prevents a potential regression where the annotation fails due to an insufficient context length.

Key Assertions:

- The function `annotate` handles the case when the context is too short and returns a fallback value.
- The function `annotate` handles the case when the context is too long and raises an error with a meaningful message.
- The function `annotate` correctly propagates the error up the call stack if it occurs due to insufficient context length.
- The function `annotate` provides a clear indication of the reason for the error (insufficient context) in its error message.
- The function `annotate` does not silently ignore the error and instead raises an exception with more information about the context.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that the OllamaProvider annotates the provided function with an error message when a call to the function raises a call error.

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

Key Assertions:

- The annotation should include the reason for failure as 'Failed after 2 retries. Last error: boom'.
- The annotation should not include any additional information about the cause of the failure.
- The annotation should only include the last error that occurred during the call to the function.
- The annotation should use the correct syntax and formatting for error messages in Python.
- The annotation should be able to handle different types of exceptions raised by the function being annotated, including but not limited to Exception.
- The annotation should not ignore or suppress any errors that occur during the execution of the function being annotated.
- The annotation should provide a clear and concise description of what went wrong when the function was called.
- The annotation should be able to handle cases where the function being annotated is not actually failing, but instead returning an error code or exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The Ollama provider should report an error when annotating a function without the required httpx dependency.

Why Needed: This test prevents potential issues where the provider incorrectly reports missing dependencies or fails to provide useful information.

Key Assertions:

- assert annotation.error == 'httpx not installed. Install with: pip install httpx'
- provider.annotate(test, 'def test_case(): assert True')
- test.nodeid == 'tests/test_sample.py::test_case'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 40-44)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_annotate_success_full_flow` 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/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the Ollama provider makes a successful API call to generate text.

Why Needed: This test prevents regression where the Ollama provider fails to make an API call due to incorrect configuration or timeout settings.

Key Assertions:

- The function `_call_ollama` of the `OllamaProvider` class returns a JSON response with the correct model and prompt.
- The URL of the API call is set correctly based on the provided host and timeout values.
- The generated text matches the expected output for the given prompt and model.
- The timeout value is respected and does not exceed the specified seconds.
- The `json` dictionary returned by the `_call_ollama` function contains all required keys (model, prompt, system, stream).
- The `timeout` value is correctly set to 60 seconds for the API 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the default model is used when not specified for Ollama provider.

Why Needed: This test prevents a regression where the default model is not used, potentially causing unexpected behavior or errors.

Key Assertions:

- The captured response from the Ollama provider contains the 'model' key with value 'llama3.2'.
- The captured response from the Ollama provider does not contain the 'model' key.
- The captured response from the Ollama provider is of type 'str', which matches the expected default model 'llama3.2'.
- The captured response from the Ollama provider contains a non-default model, which breaks the expected 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the Ollama provider returns False when the server is unavailable.

Why Needed: This test prevents a regression where the provider might return True when the server is available, potentially causing unexpected behavior in downstream applications.

Key Assertions:

- The function `_check_availability()` of the `OllamaProvider` class should return False for any URL.
- A ConnectionError exception should be raised when calling `fake_get(url)` with a valid URL.
- The provider's internal state should not change unexpectedly after setting up the mock HTTPX instance.
- The provider's `_check_availability()` method should only check if the server is available and return False in this case.
- Any other URLs passed to the `fake_get(url)` function should raise a ConnectionError exception.
- If the server is not running, the provider should still be able to detect it by checking for an empty response from the fake HTTPX instance.
- The provider's internal state should remain unchanged if the mock HTTPX instance is properly set up and configured.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: To prevent a regression where the provider incorrectly assumes all requests are successful (status code 200) when they may not be.

Key Assertions:

- assert provider._check_availability() == False
- assert FakeResponse().status_code != 200
- assert config.provider == 'ollama'
- assert isinstance(provider, OllamaProvider)
- assert hasattr(provider, '_check_availability')
- assert not callable(provider._check_availability)
- assert provider.config is not None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that the Ollama provider checks its availability via the /api/tags endpoint successfully.

Why Needed: This test prevents a potential bug where the Ollama provider does not respond to requests for tags, potentially leading to unexpected behavior or errors in downstream applications.

Key Assertions:

- The `/api/tags` endpoint should be present in any request that checks availability.
- The response from the `/api/tags` endpoint should have a status code of 200 (OK).
- The `ollama_host` attribute in the `Config` object should match the host URL of the Ollama provider (`http://localhost:11434`).

COVERAGE

src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The Ollama provider function `is_local()` returns True for local configurations.

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

Key Assertions:

- config.provider == 'ollama'
- provider.is_local() == True
- assert isinstance(provider, OllamaProvider)
- provider.config is of type Config
- provider.config.provider == 'ollama'
- provider.config.provider != 'local'

COVERAGE

src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that the `OllamaProvider` class throws an error when parsing a response with invalid JSON.

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

Key Assertions:

- annotation.error == 'Failed to parse LLM response as JSON'
- provider._parse_response('not-json') is not None
- provider._parse_response('not-json').error == 'Failed to parse LLM response as JSON'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 52-53, 186-187, 190-192)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-52, 55)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The Ollama provider rejects invalid key_assertions payloads.

Why Needed: This test prevents the provider from incorrectly handling malformed input data.

Key Assertions:

- response_data['key_assertions'] is not a list
- json.dumps(response_data) does not contain 'key_assertions'
- The value of 'key_assertions' in response_data does not match the expected format

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the provider does not extract JSON from code fences, potentially leading to incorrect or incomplete annotations.

Key Assertions:

- The response is a valid JSON object.
- The response contains all required keys (e.g. 'text', 'code').
- The response has the correct structure and formatting (e.g. double quotes around key-value pairs).
- All nested objects have the correct structure and formatting (e.g. arrays of strings, dictionaries with string keys).
- The provider correctly handles nested code blocks (e.g. multi-line strings, code snippets).

COVERAGE

src/pytest_llm_report/collector.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: 413-414, 417, 421-423)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_parse_response_json_in_plain_fence` 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the Ollama provider correctly parses valid JSON responses.

Why Needed: Prevents regressions due to changes in response structure or content.

Key Assertions:

- assert a is not None
- assert b is not None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

📄 tests/test_models.py

29 tests

PASSED `tests/test_models.py::TestArtifactEntry::test_to_dict` 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	4 lines (ranges: 260-263)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where the coverage data is not properly serialized to JSON.

Key Assertions:

- The 'file_path' key in the dictionary should match the expected value.
- The 'line_ranges' key in the dictionary should match the expected value.
- The 'line_count' key in the dictionary should match the expected value.
- The 'coverage_data' key (if any) should be an empty list or None, as coverage data is not serialized to JSON.
- Any additional keys in the dictionary should have values that are consistent with the original CoverageEntry object.
- The 'file_path', 'line_ranges', and 'line_count' fields should match their respective attributes of the CoverageEntry class.
- If a CoverageEntry object has additional data (e.g., coverage data), it should be preserved 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/models.py	3 lines (ranges: 213-215)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Tests CoverageEntry serialization correctly.

Why Needed: CoverageEntry should be able to serialize its internal state accurately.

Key Assertions:

- assert d['file_path'] == 'src/foo.py',
- assert d['line_ranges'] == '1-3, 5, 10-15',
- assert d['line_count'] == 10

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`4 lines (ranges: 40-43)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where an empty annotation would not have any confidence or error.

Key Assertions:

- annotation.scenario == "" (empty string)
- annotation.why_needed == "" (default value for confidence and error)
- annotation.key_assertions == [] (no key assertions are performed on an empty annotation)
- assert annotation.confidence is None (expected None for default confidence)
- assert annotation.error is None (expected None for default 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: 413-414, 417, 421-423)`

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 the required fields.

Why Needed: This test prevents a potential bug where the minimal annotation is missing some of its necessary fields.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	8 lines (ranges: 104-107, 109, 111, 113, 115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where only partial information is returned.

Key Assertions:

- The 'scenario' key should contain the correct value.
- The 'confidence' key should contain the expected value.
- The 'context_summary' key should have the correct mode and bytes values.
- The 'mode' key in the context summary should be set to 'minimal'.
- The 'bytes' key in the context summary should be set to 1000 or more.

COVERAGE

src/pytest_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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test default report schema version and empty lists.

Why Needed: Prevents a potential bug where the default report might be missing required fields.

Key Assertions:

- The 'schema_version' field in the test report should match the expected value of SCHEMA_VERSION.
- The 'tests' field in the test report should be an empty list.
- The 'warnings' field in the test report should not be present (excluded).
- The 'collection_errors' field in the test report should not be present (excluded).

COVERAGE

src/pytest_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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test Report with Collection Errors should include them.

Why Needed: This test prevents a regression where the report does not accurately identify collection errors.

Key Assertions:

- The length of `collection_errors` is 1.
- The value of `nodeid` in the first error is 'test_bad.py'.
- Each error in `collection_errors` has a unique `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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test reports the presence of warnings in a ReportRoot object.

Why Needed: This test prevents a regression where warnings are not reported correctly.

Key Assertions:

- The length of the 'warnings' list should be 1.
- The code of the first warning should match 'W001'.
- Each warning in the 'warnings' list should have a 'code' attribute matching '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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Tests should be sorted by nodeid in output.

Why Needed: This test prevents regression where the order of tests is not guaranteed to be consistent due to a bug in sorting algorithms or data structures.

Key Assertions:

- The list of nodeids returned from `to_dict()` matches the expected list.
- Each nodeid in the list corresponds to a test result with an outcome of 'passed'.
- No duplicate nodeids are present in the list.
- All nodeids are unique and do not contain any duplicates.
- The order of nodeids is consistent across different runs of the test.
- No tests have been skipped or ignored during sorting.
- Each nodeid has a corresponding test result with an outcome of 'passed'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`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: 413-414, 417, 421-423)`

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

AI ASSESSMENT

Scenario: Test 'test_to_dict_with_detail' verifies that a ReportWarning instance can be converted to a dictionary with the required keys.

Why Needed: This test prevents a potential bug where the detailed warning message is not included in the dictionary representation of the ReportWarning instance.

Key Assertions:

- The value of 'detail' in the dictionary representation of the ReportWarning instance should be '/path/to/file'.
- The value of 'code' in the dictionary representation of the ReportWarning instance should be 'W001'.
- The value of 'message' in the dictionary representation of the ReportWarning instance should be 'No coverage'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	6 lines (ranges: 235-237, 239-241)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_models.py::TestReportWarning::test_to_dict_without_detail` 1ms 3

AI ASSESSMENT

Scenario: Test 'test_to_dict_without_detail' verifies that a ReportWarning object is created without detail.

Why Needed: This test prevents the creation of a warning with no detailed message, which could lead to unexpected behavior or errors.

Key Assertions:

- The 'detail' key should be excluded from the warning dictionary.
- The warning code should match 'W001'.
- The warning message should not include any details.
- A detail message should not be present in the warning dictionary.
- The warning object should have a non-empty 'code' attribute.
- The warning object should have a non-empty 'message' attribute.
- The warning object should not have a 'detail' 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	5 lines (ranges: 235-237, 239, 241)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that RunMeta has aggregation fields.

Why Needed: This test prevents regression where RunMeta is missing aggregation fields, potentially leading to incorrect data analysis.

Key Assertions:

- The 'run_id' key in the run metadata should be equal to 'run-123'.
- The 'run_group_id' key in the run metadata should be equal to 'group-456'.
- The 'is_aggregated' key in the run metadata should be True.
- The 'aggregation_policy' key in the run metadata should be set to 'merge'.
- The 'run_count' key in the run metadata should be equal to 3.
- The length of the 'source_reports' list in the run metadata should be equal to 2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	39 lines (ranges: 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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that LLM fields are excluded when annotations are disabled.

Why Needed: Prevents regression where LLMs are enabled but annotations are disabled.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that LLM traceability fields are included when enabled for the specified model.

Why Needed: This test prevents a regression where the llm_traceability_fields attribute is missing or incorrectly set in RunMeta instances.

Key Assertions:

- The value of llm_annotations_enabled should be True.
- The value of llm_provider should match 'ollama'.
- The value of llm_model should match 'llama3.2:1b'.
- The value of llm_context_mode should match 'complete'.
- The value of llm_annotations_count should be 10.
- The value of llm_annotations_errors 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	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: 413-414, 417, 421-423)

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 the `non_aggregated_excludes_source_reports` method does not include `source_reports` in its aggregated reports.

Why Needed: This test prevents a regression where non-aggregated reports are incorrectly including source reports, which can lead to confusion and incorrect analysis.

Key Assertions:

- The 'source_reports' key is not present in the report dictionary.
- The value of `is_aggregated` is set to `False` for this report.
- The presence of `source_reports` in the report dictionary would indicate an aggregated report, which is unexpected.

COVERAGE

`src/pytest_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: 413-414, 417, 421-423)`

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

AI ASSESSMENT

Scenario: Test RunMeta to dict with all optional fields.

Why Needed: Prevents regression in case of missing or invalid metadata.

Key Assertions:

- Verify that the 'git_sha' field is set correctly.
- Check if the 'git_dirty' field is True for source reports.
- Assert that the 'repo_version', 'repo_git_sha', and 'plugin_git_sha' fields are set correctly.
- Verify that the 'config_hash' field is set correctly.
- Count the number of source report in the output data.
- Check if all required fields are present in the output 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`	`49 lines (ranges: 283-285, 287-289, 370-386, 388-411, 413, 425)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

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

AI ASSESSMENT

Scenario: TestRunMeta::test_run_status_fields verifies that RunMeta includes run status fields.

Why Needed: This test prevents a regression where the RunMeta object is missing certain required fields.

Key Assertions:

- The 'exit_code' field should be set to 1.
- The 'interrupted' field should be True.
- The 'collect_only' field should be True.
- The 'collected_count' field should equal 10.
- The 'selected_count' field should equal 8.
- The 'deselected_count' field should equal 2.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: Prevents regression where the schema version is incorrectly formatted or missing.

Key Assertions:

- The schema version should be split into three parts (e.g., '1.2.3').
- Each part of the schema version should be a digit (0-9).
- All parts of the schema version should be separated by dots (.), not colons (:).

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

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

AI ASSESSMENT

Scenario: Test that the `ReportRoot` class includes the schema version in its report root.

Why Needed: This test prevents a potential bug where the schema version is not included in the report root, potentially leading to incorrect analysis or reporting.

Key Assertions:

- The `schema_version` attribute of the `ReportRoot` object should be equal to `SCHEMA_VERSION`.
- The `to_dict()` method of the `ReportRoot` object should return a dictionary with a `schema_version` key that equals `SCHEMA_VERSION`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	54 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Testing the `CoverageEntry` class to ensure it correctly serializes a coverage entry.

Why Needed: This test prevents regression where the coverage entry is not properly serialized, potentially leading to incorrect or missing information in the output.

Key Assertions:

- The 'file_path' key should match the expected value of 'src/foo.py'.
- The 'line_ranges' key should contain the correct ranges and count.
- The 'line_count' key should have the expected value of 10.
- Any additional keys in the output dictionary should be empty or None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	8 lines (ranges: 71-78)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: The test verifies that the 'to_dict' method of LlmAnnotation returns a dictionary with required fields.

Why Needed: This test prevents regression by ensuring that the minimal annotation format is correctly serialized.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	5 lines (ranges: 283-285, 287, 289)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test SourceReport to_dict_with_run_id method with a SourceReport object.

Why Needed: This test prevents a potential bug where the run ID is not included in the dictionary representation of a SourceReport.

Key Assertions:

- The 'run_id' key should be present in the dictionary representation of a SourceReport.
- The value of the 'run_id' key should match the provided run_id.
- The 'sha256' and 'path' keys should not affect the presence or value of the 'run_id' 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	6 lines (ranges: 283-285, 287-289)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where coverage data is not properly serialized to JSON.

Key Assertions:

- The 'file_path' key in the dictionary should match the expected value.
- The 'line_ranges' key in the dictionary should match the expected value.
- The 'line_count' key in the dictionary should match the expected value.
- The 'coverage_data' key is not present in the dictionary (this test is actually testing `CoverageEntry.to_dict()` without any coverage data).
- The values of 'file_path', 'line_ranges', and 'line_count' are correct.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	11 lines (ranges: 455-463, 465, 467)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED tests/test_models.py::TestTestCaseResult::test_minimal_result 1ms 3

AI ASSESSMENT

Scenario: Test that the `minimal_result` object has all required fields.

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

Key Assertions:

- The value of `nodeid` should be set to `test_foo.py::test_bar`.
- The value of `outcome` should be set to `passed`.
- The value of `duration` should be set to `0.0`.
- The value of `phase` 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: 413-414, 417, 421-423)

PASSED tests/test_models.py::TestTestCaseResult::test_result_with_coverage 1ms 3

AI ASSESSMENT

Scenario: Test verifies that the `result` dictionary includes a single 'coverage' key with a list of coverage entries.

Why Needed: This test prevents regression in case the coverage report is missing or incorrect.

Key Assertions:

- The 'coverage' key should exist in the 'result' dictionary.
- The 'coverage' key should contain a list of coverage entries.
- Each coverage entry should have 'file_path' and 'line_ranges' keys.
- All file paths in the 'coverage' list should match the expected source code.
- All line ranges in the 'coverage' list should be between 1 and 5 (inclusive).
- The total number of coverage entries should be exactly 1.
- Each coverage entry's 'file_path' value should match the expected file path ('src/foo.py').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test Result with LLM Opt-Out should include flag.

Why Needed: To prevent regression when LLM opt-out is enabled.

Key Assertions:

- The 'llm_opt_out' key in the result dictionary should be set to True.
- The 'llm_opt_out' value in the result dictionary should be a boolean.
- When LLM Opt-Out is enabled, the 'llm_opt_out' flag should be present in the result.
- When LLM Opt-Out is disabled, the 'llm_opt_out' flag should not be present in the result.
- The presence of the 'llm_opt_out' key and value should ensure consistency across different test cases.
- The absence of the 'llm_opt_out' key or value should indicate a regression issue when LLM Opt-Out 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	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: 413-414, 417, 421-423)

PASSED tests/test_models.py::TestTestCaseResult::test_result_with_rerun 1ms 3

AI ASSESSMENT

Scenario: Test 'Result with reruns' verifies that the 'rerun_count' and 'final_outcome' fields are correctly populated in the TestCaseResult object.

Why Needed: This test prevents regression where a result is not properly updated when rerunning tests.

Key Assertions:

- The 'rerun_count' field should be equal to the expected value of 2.
- The 'final_outcome' field should always be set to 'passed'.
- The 'rerun_count' and 'final_outcome' fields should not be empty or None when a result is created.

COVERAGE

src/pytest_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: 413-414, 417, 421-423)

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the result of a test is rerunned, potentially hiding important information about its outcome.

Key Assertions:

- The 'rerun_count' key should be absent from the `result` dictionary.
- The 'final_outcome' key should not exist in the `result` dictionary.
- The 'rerun_count' and 'final_outcome' keys should not appear in the `result` dictionary.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`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: 413-414, 417, 421-423)`

📄 tests/test_options.py

16 tests

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

AI ASSESSMENT

Scenario: Test that default values are set correctly.

Why Needed: Prevents a bug where the default configuration settings are not properly initialized, potentially leading to unexpected behavior or errors in subsequent tests.

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verify that `get_default_config()` returns a default configuration with no provider.

Why Needed: Prevents regression when using `Config` without a provider.

Key Assertions:

- The function `get_default_config()` should return an instance of `Config`.
- The `cfg.provider` attribute should be set to `'none'`.
- The configuration should have no provider specified.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test that the `is_llm_enabled` check returns False for a provider without an LLM.

Why Needed: Prevents regression in case of a change to the `provider` parameter.

Key Assertions:

- The function `Config.is_llm_enabled()` should return `False` when the `provider` is set to `''none''`.
- The function `Config.is_llm_enabled()` should return `True` when the `provider` is set to `''ollama''`.
- The function `Config.is_llm_enabled()` should not return any value (i.e., raise an exception) when the `provider` is set to `''none''`.
- The function `Config.is_llm_enabled()` should return `False` when the `provider` is set to `''ollama''` and then changed back to `''none''`.
- The function `Config.is_llm_enabled()` should not raise an exception when the `provider` is set to `''ollama''`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 252)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Testing the validation of an invalid context mode for the llm_context_mode setting.

Why Needed: This test prevents a potential bug where the llm_context_mode is set to a value that is not supported by the model.

Key Assertions:

- The configuration object should have exactly one error message related to the 'Invalid llm_context_mode' context mode.
- The error message should contain the string 'Invalid llm_context_mode 'mega_max'.
- The test should fail if more than one error message is found in the configuration object.
- The error messages should be present in the list of errors returned by the validate() method.
- The first error message should be a string indicating that the context mode is invalid.
- The 'Invalid llm_context_mode' context mode should not be supported by the 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/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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test validates configuration with an invalid include phase.

Why Needed: Prevents a potential bug where the validation of an invalid include phase is not caught.

Key Assertions:

- Check that the `validate()` method returns exactly one error message for an invalid include phase.
- Verify that the error message contains the specified 'Invalid include_phase' phrase.
- Ensure that the error message does not contain any other relevant information about 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	22 lines (ranges: 123, 163, 191, 194-197, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Test validation of numeric constraints for TestConfig.

Why Needed: Prevents regression where the llm_context_bytes is set to a value less than 1000, potentially causing issues with LLM context creation.

Key Assertions:

- cfg.validate() returns an error message indicating that llm_context_bytes must be at least 1000
- llm_context_bytes is not >= 1000 in the errors list
- llm_max_tests should be positive or 0 to prevent invalid configuration
- llm_requests_per_minute should be at least 1 to prevent invalid configuration
- llm_timeout_seconds should be at least 1 to prevent invalid configuration
- llm_max_retries should be 0 or positive to prevent invalid 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	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: 413-414, 417, 421-423)

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

AI ASSESSMENT

Scenario: Verifies that the `validate()` method returns an empty list for a valid configuration.

Why Needed: Prevents a potential bug where an invalid configuration causes the validation to fail without raising any errors.

Key Assertions:

- The `validate()` method is called with an empty configuration object.
- An error message indicating that no errors were found is not raised.
- The function does not throw any exceptions or raise informative error messages.
- The test verifies the absence of any validation errors.
- The test ensures that the function behaves as expected for 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/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: 413-414, 417, 421-423)

PASSED tests/test_options.py::TestLoadConfig::test_load_aggregation_options 1ms 3

AI ASSESSMENT

Scenario: Test the `load_aggregation_options` function to ensure it correctly loads aggregation options from a mock Pytest configuration.

Why Needed: This test prevents regression in the `load_aggregation_options` function, which is responsible for loading aggregation options from a Pytest configuration. Without this test, the function may incorrectly load options or fail to load certain options, leading to unexpected behavior or errors.

Key Assertions:

- The `aggregate_dir` attribute of the loaded configuration should be set to 'aggr_dir'.
- The `aggregate_policy` attribute of the loaded configuration should be set to 'merge'.
- The `aggregate_run_id` attribute of the loaded configuration should be set to 'run-123'.
- The `aggregate_group_id` attribute of the loaded configuration should be set to 'group-abc'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	33 lines (ranges: 123, 163, 276, 279, 281, 283, 285, 287, 289, 291, 298, 305-307, 310, 319, 321, 323, 325, 327, 329, 331, 335-343, 347, 349)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options.py::TestLoadConfig::test_load_config_invalid_int_ini` 1ms 3

AI ASSESSMENT

Scenario: Test Load Config: Handling of Invalid Integer Values in INI.

Why Needed: Prevents a potential crash when encountering invalid integer values in the INI file.

Key Assertions:

- The function `load_config` should not crash or throw an exception when it encounters an 'llm_report_max_retries' value of 'garbage'.
- The function `load_config` should return a valid configuration object with the default value for 'llm_max_retries' (10) if no valid integer value is found in the INI file.
- The function `load_config` should not raise an exception when it encounters an invalid integer value in the INI file, but instead handle it by returning a fallback value or crashing 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/options.py	33 lines (ranges: 123, 163, 276, 279, 281, 283, 285, 287, 289, 291-295, 298, 305-307, 310, 319, 321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED tests/test_options.py::TestLoadConfig::test_load_coverage_source 1ms 3

AI ASSESSMENT

Scenario: Verify that the `llm_coverage_source` option is set to 'cov_dir' when loading configuration.

Why Needed: This test prevents a potential bug where the coverage source is not correctly set, leading to incorrect coverage analysis.

Key Assertions:

- The value of `llm_coverage_source` in the loaded configuration is indeed 'cov_dir'.
- The `llm_coverage_source` option is correctly set to 'cov_dir' when loading the configuration.
- The coverage source is not set to an empty string or None, as it would be if there was no coverage source specified.
- The value of `llm_coverage_source` in the loaded configuration matches the expected value of 'cov_dir'.
- The `llm_coverage_source` option is correctly set to a valid directory path.
- The test does not fail when an empty string or None is passed as the coverage source.
- The test passes with all assertions passing for the given scenario.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	30 lines (ranges: 123, 163, 276, 279, 281, 283, 285, 287, 289, 291, 298, 305-307, 310, 319, 321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343-344, 347, 349)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED tests/test_options.py::TestLoadConfig::test_load_defaults 1ms 3

AI ASSESSMENT

Scenario: Testing the `load_defaults` function with default configuration.

Why Needed: Prevents a potential bug where the 'report_html' option is not set when no options are provided.

Key Assertions:

- The `cfg.provider` attribute should be set to 'none'.
- The `cfg.report_html` attribute should be None.
- The function does not raise an exception if no configuration options are provided.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	29 lines (ranges: 123, 163, 276, 279, 281, 283, 285, 287, 289, 291, 298, 305-307, 310, 319, 321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options.py::TestLoadConfig::test_load_from_cli_overrides_ini` 1ms 3

AI ASSESSMENT

Scenario: Test that CLI options override ini options when loading configuration from command line.

Why Needed: This test prevents a regression where the default value of `llm_report_html` is not overridden by CLI options.

Key Assertions:

- The `report_html` option in the configuration should be set to 'cli_report.html'.
- The `llm_requests_per_minute` option in the configuration should be set to 100.
- The value of `llm_report_html` option should not be overridden by CLI options.
- The value of `llm_requests_per_minute` option should override any ini 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	32 lines (ranges: 123, 163, 276, 279, 281, 283, 285, 287-289, 291, 298, 305-307, 310, 319-321, 323, 325, 327, 329-331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED tests/test_options.py::TestLoadConfig::test_load_from_cli_retries 1ms 3

AI ASSESSMENT

Scenario: Testing the `load_from_cli` method with a retry limit of 2.

Why Needed: This test prevents a potential bug where the `llm_max_retries` option is not properly set, causing the configuration to be loaded without retries.

Key Assertions:

- The value of `llm_max_retries` should be equal to 2.
- The `load_from_cli` method should be able to retrieve the configuration with a retry limit of 2.
- The `llm_max_retries` option should not cause any issues during configuration loading.
- The test should fail if the `llm_max_retries` option is set to a value other than 1 or 2.
- The `load_from_cli` method should be able to handle cases where retries are required but not specified.
- The `load_from_cli` method should be able to retry loading the configuration multiple times if necessary.
- The test should pass if the `llm_max_retries` option is set to a value that allows for retries (e.g., 3 or more).

COVERAGE

src/pytest_llm_report/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, 281, 283, 285, 287, 289, 291, 298, 305-307, 310, 319, 321, 323, 325, 327, 329, 331-332, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED tests/test_options.py::TestLoadConfig::test_load_from_ini 1ms 3

AI ASSESSMENT

Scenario: Test loading values from ini options to ensure correct configuration retrieval.

Why Needed: This test prevents a potential bug where the `load_config` function returns incorrect or missing configuration values due to an issue with the `getini` method's side effect.

Key Assertions:

- The `provider` key in the configuration should be set to 'ollama'.
- The `model` key in the configuration should be set to 'llama3'.
- The `llm_context_mode` key in the configuration should be set to 'balanced'.
- The `llm_requests_per_minute` key in the configuration should be set to 10.
- The `llm_max_retries` key in the configuration should be set to 2.
- The `report_html` key in the configuration should be set to 'report.html'.
- The `report_json` key in the configuration should be set to 'report.json'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	37 lines (ranges: 123, 163, 276, 279-293, 298, 305-307, 310, 319, 321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

📄 tests/test_options_extended.py

13 tests

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_aggregation_settings` 1ms 3

AI ASSESSMENT

Scenario: Tests the aggregation settings configuration.

Why Needed: Prevents a potential bug where the aggregate directory, policy, and history inclusion are not correctly set for aggregated data.

Key Assertions:

- The `aggregate_dir` attribute is set to `/reports` as expected.
- The `aggregate_policy` attribute is set to `merge` as expected.
- The `aggregate_include_history` attribute is set to `True` as expected.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_all_output_paths` 1ms 3

AI ASSESSMENT

Scenario: Test Config with all output paths.

Why Needed: Prevents regression in case of multiple report formats being used.

Key Assertions:

- The value of `config.report_html` is set to `report.html`.
- The value of `config.report_json` is set to `report.json`.
- The value of `config.report_pdf` is set to `report.pdf`.
- The expected values for `config.report_evidence_bundle`, `config.report_dependency_snapshot` are not set.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_capture_settings` 1ms 3

AI ASSESSMENT

Scenario: Test the `capture_failed_output` configuration option.

Why Needed: This test prevents a potential bug where the captured output exceeds the maximum allowed length of 8000 characters.

Key Assertions:

- config.capture_failed_output is True
- assert config.capture_failed_output is True
- The captured output does not exceed 8000 characters

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_compliance_settings` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `Config` object is created with specified compliance settings.

Why Needed: This test prevents a potential bug where the `Config` object's metadata and HMAC key files are not set correctly, leading to incorrect configuration.

Key Assertions:

- The `metadata_file` attribute of the `Config` object should be set to 'metadata.json'.
- The `hmac_key_file` attribute of the `Config` object should be set to 'key.txt'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_coverage_settings` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `Config` class correctly sets `omit_tests_from_coverage` to `False` and `include_phase` to `''all''`.

Why Needed: Prevents a potential regression where coverage settings are not properly configured.

Key Assertions:

- config.omit_tests_from_coverage is set to `False`
- config.include_phase is set to `''all''`
- The configuration does not include any tests from the specified 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/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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 configuration.

Why Needed: This test prevents a potential bug where the default exclude globs are not properly handled when custom ones are provided.

Key Assertions:

- The '*.pyc' and '*.log' exclude globs should be included in the config.
- The custom exclude glob '*.pyc' should match one of the existing exclude globs.
- The custom exclude glob '*.log' should also match one of the existing exclude globs.
- The include glob '*' should not be present in the config.
- The custom exclude globs should override any default exclude globs if provided.
- The custom exclude glob '*.pyc' should only match files with a .pyc extension.
- The custom exclude glob '*.log' should only match files with a .log extension.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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 all specified globs.

Why Needed: This test prevents a potential bug where the include globs are not correctly propagated to the LLM context.

Key Assertions:

- The `*.py` glob matches any Python files in the current directory and subdirectories.
- The `*.pyi` glob matches any Python source file with an `.i` extension.
- The `*.py` glob is included in the list of include globs for the LLM context.
- The `*.pyi` glob is included in the list of include globs for the LLM context.
- The configuration option `llm_context_include_globs` is correctly set to `['*.py', '*.pyi']`.
- No other include globs are added to the configuration option `llm_context_include_globs`.
- The include globs are correctly propagated to the LLM context.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_invocation_settings` 1ms 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 123)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_llm_execution_settings` 1ms 3

AI ASSESSMENT

Scenario: Test the LLM execution settings configuration.

Why Needed: Prevents regression in LLM execution settings when using multiple tests concurrently or with high concurrency.

Key Assertions:

- The value of llm_max_tests is set to 50.
- The value of llm_max_concurrency is set to 8.
- The value of llm_requests_per_minute is set to 12.
- The value of llm_timeout_seconds is set to 60 seconds.
- The value of llm_cache_ttl_seconds is set to 3600 seconds (1 hour).
- The cache directory is set to .cache.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_llm_param_settings` 1ms 3

AI ASSESSMENT

Scenario: Test the configuration of LLM parameter settings.

Why Needed: This test prevents a potential bug where the LLM include param values are not correctly set to True, potentially leading to incorrect output or errors.

Key Assertions:

- config.llm_include_param_values is True
- config.llm_param_value_max_chars == 200

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_llm_settings` 1ms 3

AI ASSESSMENT

Scenario: Verify that the LLM settings are correctly configured with the provided provider, model, and context settings.

Why Needed: This test prevents a potential bug where the LLM settings are not properly set, potentially leading to incorrect results or errors during training.

Key Assertions:

- The `provider` attribute is set to 'ollama'.
- The `model` attribute is set to 'llama3.2'.
- The `llm_context_bytes` attribute is set to 64000.

COVERAGE

src/pytest_llm_report/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: 413-414, 417, 421-423)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_repo_root_path` 1ms 3

AI ASSESSMENT

Scenario: Verify that the `repo_root` attribute of a `TestConfigAnnotations` instance is correctly set to the specified path.

Why Needed: This test prevents a potential bug where the `repo_root` attribute is not set correctly, potentially leading to incorrect configuration or unexpected behavior.

Key Assertions:

- config.repo_root is an instance of `Path` and its value matches the expected path `/project`
- config.repo_root.path is equal to '/project'
- config.repo_root.parent is None (i.e., it's not a subdirectory)
- config.repo_root.is_absolute() is True
- config.repo_root.exists() is False
- config.repo_root.is_dir() is False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_valid_phase_values` 1ms 3

AI ASSESSMENT

Scenario: Test the `test_valid_phase_values` function to ensure all valid include_phase values pass validation.

Why Needed: Prevents a potential bug where invalid or missing include_phase values cause test failures.

Key Assertions:

- All phases are included in the configuration.
- No phase is excluded from the configuration.
- The `include_phase` value for each phase is present and valid.
- Invalid or missing include_phase values are not included in the errors.
- The validation process does not return any errors for all valid include_phase values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	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: 413-414, 417, 421-423)

📄 tests/test_options_maximal.py

10 tests

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_exclude_globs` 1ms 3

AI ASSESSMENT

Scenario: Verify that the default exclude globs are correctly set for the LLM context.

Why Needed: This test prevents a potential bug where the default exclude globs do not include all necessary files.

Key Assertions:

- The function `*.pyc` should be included in the default exclude globs.
- The function `__pycache__/*` should be included in the default exclude globs.
- The function `*secret*` should be included in the default exclude globs.
- The function `*password*` should be included in the default exclude globs.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_redact_patterns` 1ms 3

AI ASSESSMENT

Scenario: Verify that default redact patterns include sensitive information.

Why Needed: This test prevents a potential security vulnerability where sensitive information like passwords and tokens are not properly redacted.

Key Assertions:

- Any pattern containing '--password' should be present in the default redact patterns.
- Any pattern containing '--token' should be present in the default redact patterns.
- Any pattern containing '--api[_-]?key' should be present in the default redact patterns.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_values` 1ms 3

AI ASSESSMENT

Scenario: Test default values of the `get_default_config()` function.

Why Needed: This test prevents a potential bug where the default values are not correctly set for the configuration.

Key Assertions:

- The value of `config.provider` should be 'none'.
- The value of `config.llm_context_mode` should be 'minimal'.
- The value of `config.llm_context_bytes` should be 32000.
- The value of `config.omit_tests_from_coverage` should be True.
- The value of `config.include_phase` should be 'run'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 123, 163, 261)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED tests/test_options_maximal.py::TestConfigHelpersMaximal::test_is_llm_enabled 1ms 3

AI ASSESSMENT

Scenario: Test that the `is_llm_enabled` method returns correct enabled status for different providers.

Why Needed: This test prevents regression in cases where LLM is not available or is disabled.

Key Assertions:

- The function should return False when provider is 'none'.
- The function should return True when provider is either '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: 413-414, 417, 421-423)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_aggregate_policy` 1ms 3

AI ASSESSMENT

Scenario: Test validates an invalid aggregate policy.

Why Needed: Prevents a potential bug where the validate method returns multiple errors for an invalid aggregate policy.

Key Assertions:

- The test verifies that the validate method returns exactly one error message.
- The error message contains the string 'Invalid aggregate_policy 'invalid''.
- The error message is not empty.
- The error message does not contain any additional information (e.g., a stack trace).

COVERAGE

src/pytest_llm_report/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: 413-414, 417, 421-423)

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: 413-414, 417, 421-423)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_include_phase` 1ms 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	23 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-221, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_provider` 1ms 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	22 lines (ranges: 123, 163, 191, 194-197, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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 when the `llm_context_bytes` value is set to a large negative number.

Key Assertions:

- The function `validate()` should return at least 4 errors for invalid numeric values.
- One error should contain 'llm_context_bytes' as a substring.
- Another error should contain 'llm_max_tests' as a substring.
- Yet another error should contain 'llm_requests_per_minute' as a substring.
- A fourth error should contain 'llm_timeout_seconds' as a substring.

COVERAGE

src/pytest_llm_report/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: 413-414, 417, 421-423)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_valid_config` 1ms 3

AI ASSESSMENT

Scenario: Verifies that an invalid configuration returns an empty list.

Why Needed: Prevents a potential bug where an invalid configuration is not properly validated and could cause unexpected behavior or errors in downstream code.

Key Assertions:

- The `validate()` method of the `Config` object should return an empty list for a valid configuration.
- A valid configuration should not produce any output when passed to the `validate()` method.
- An invalid configuration should raise an exception or return an error message indicating that it is invalid.
- The `validate()` method should be able to handle a wide range of input configurations without crashing or producing unexpected results.
- The `validate()` method should not modify the original configuration object in any way.
- A valid configuration should have no side effects on the system.
- An invalid configuration should cause an exception to be raised with a clear and descriptive 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	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: 413-414, 417, 421-423)

📄 tests/test_plugin_integration.py

6 tests

PASSED `tests/test_plugin_integration.py::TestPluginConfigLoading::test_config_defaults` 1ms 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 of type `Config`.
- The `cfg` variable should not contain any explicitly registered pytest options.
- The `cfg` variable's values should match the expected safe defaults for the plugin 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	42 lines (ranges: 123, 163, 276, 279-287, 289, 291-293, 298, 305-308, 310-313, 319, 321-325, 327, 329, 331, 335, 337, 339-341, 343, 347, 349)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_plugin_integration.py::TestPluginConfigLoading::test_markers_exist_in_config` 1ms 2

AI ASSESSMENT

Scenario: Verify that the `pytestconfig` object is accessible in the test.

Why Needed: This test prevents a potential issue where the plugin configuration is not available due to an incorrect or missing configuration.

Key Assertions:

- pytestconfig is not None
- pytestconfig has attributes and methods (e.g., `__dict__`, `config`) that are accessible

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_plugin_integration.py::TestPluginIntegration::test_llm_context_marker` 1ms 2

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

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: 413-414, 417, 421-423)

PASSED `tests/test_plugin_integration.py::TestPluginIntegration::test_requirement_marker` 1ms 2

AI ASSESSMENT

Scenario: The `requirement_marker` function is tested to ensure it does not throw any errors when used.

Why Needed: This test prevents a potential bug where the `requirement_marker` function could cause an error due to incorrect usage.

Key Assertions:

- The `requirement_marker` function should be called with no arguments.
- The `requirement_marker` function should not throw any exceptions when called without arguments.
- The `requirement_marker` function should not raise any errors when called without 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`	`6 lines (ranges: 413-414, 417, 421-423)`

PASSED `tests/test_plugin_integration.py::TestReportGeneration::test_report_writer_integration` 36ms 6

AI ASSESSMENT

Scenario: Test the integration of report writer with pytest_llm_report.

Why Needed: This test prevents regression that may occur when the report writer is integrated into pytest_llm_report.

Key Assertions:

- Verify that a full report is generated and contains both JSON and HTML files.
- Check if the total number of tests passed is correct (1 passed, 1 failed).
- Ensure the test names 'test_a.py' and 'test_b.py' are included in the report.
- Verify the presence of both JSON and HTML files in the generated report.
- Confirm that the report contains a summary with total count of tests (2 total tests, 1 passed).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: 413-414, 417, 421-423)
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` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that collectreport skips when the plugin is disabled.

Why Needed: To prevent a regression where collectreport fails to run due to the plugin being disabled.

Key Assertions:

- mock_report.session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_collectreport.mock_report.session.config.stash.get.return_value is False
- mock_report.session.config.stash.get.assert_not_called()
- pytest_collectreport.mock_report.session.config.stash.get.return_value is None
- _enabled_key in pytest_collectreport.mock_report.session.config.stash.get.call_args_list
- pytest_collectreport.mock_report.session.config.stash.get.call_args_list[0][1] is False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	10 lines (ranges: 413-414, 417, 421-423, 434-435, 441-442)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_enabled` 2ms 2

AI ASSESSMENT

Scenario: Verify that collectreport is called when pytest_collectreport is enabled.

Why Needed: This test prevents a regression where the collector is not called when pytest_collectreport is enabled.

Key Assertions:

- The `pytest_collectreport` function should be called with the `mock_report` object as its argument.
- The `handle_collection_report` method of the `mock_collector` object should be called once with the `mock_report` object as its argument.
- The `stash_get` method of the `mock_report.session.config.stash` object should return `True` when the `_enabled_key` is present in the stash.
- The `stash_get` method of the `mock_report.session.config.stash` object should not return `True` when the `_collector_key` is present in the stash.
- The `handle_collection_report` method of the `mock_collector` object should be called with the `mock_report` object as its argument, even if it's not collecting a report.
- The `stash_get` method of the `mock_report.session.config.stash` object should return `None` when the `_enabled_key` is not present in the stash.
- The `stash_get` method of the `mock_report.session.config.stash` object should return `True` when the `_collector_key` is not present in the stash.
- The `handle_collection_report` method of the `mock_collector` object should be called with the `mock_report` object as its argument, even if it's not collecting a 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	12 lines (ranges: 413-414, 417, 421-423, 434-435, 441, 445-447)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_no_session` 1ms 2

AI ASSESSMENT

Scenario: Verify that `pytest_collectreport` does not raise an exception when a mock report object is created without a valid session.

Why Needed: This test prevents a potential bug where the plugin would attempt to access the `session` attribute of a mock report object, potentially causing unexpected behavior or errors.

Key Assertions:

- The `pytest_collectreport` function does not raise an exception when called with a mock report object that lacks a valid session.
- The `pytest_collectreport` function ignores the `session` attribute of the mock report object.
- The `pytest_collectreport` function continues to execute without interruption even if it encounters a mock report object without a valid session.
- The plugin does not attempt to access or manipulate the `session` attribute of the mock report object.
- The test case passes successfully even when the `pytest_collectreport` function is called with an invalid mock report object.
- The plugin remains unaffected by the absence of a session attribute in the mock report object.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	8 lines (ranges: 413-414, 417, 421-423, 434, 438)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_session_none` 1ms 2

AI ASSESSMENT

Scenario: Verify that `pytest_collectreport` does not raise an exception when the session is set to `None`.

Why Needed: To prevent a potential bug where `pytest_collectreport` raises an error when given a `None` session.

Key Assertions:

- The function `pytest_collectreport` should not be called with a `None` argument.
- A `None` value for the `session` attribute in the mock report object should not cause an exception to be raised.
- The `pytest_collectreport` function should continue executing without error when given a `None` session.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`8 lines (ranges: 413-414, 417, 421-423, 434, 438)`

PASSED tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_llm_enabled_warning 3ms 3

AI ASSESSMENT

Scenario: Test that LLM enabled warning is raised when configuring pytest_llm_report plugin.

Why Needed: Prevents a potential bug where the LLM report provider 'ollama' is enabled without proper configuration.

Key Assertions:

- Mocking `pytest_configure` with a valid config that includes `llm_report_provider='ollama'` and setting `llm_report_html`, `llm_report_json`, etc. to None should not trigger the warning.
- The `option.llm_report_html` is set to `None` but it's still possible for the warning to be raised if the user doesn't configure the LLM report provider correctly.
- If `llm_report_json` or any other option that depends on `llm_report_provider` is not configured, setting it to `None` should prevent the warning from being raised.
- The `option.llm_evidence_bundle`, `option.llm_dependency_snapshot`, etc. options are not affected by the LLM report provider configuration and can be set to any value without triggering the warning.
- The `option.llm_aggregate_dir`, `option.llm_aggregate_policy`, `option.llm_aggregate_run_id`, `option.llm_aggregate_group_id` options are also not affected by the LLM report provider configuration and can be set to any value without triggering the warning.
- If `llm_max_retries` is set, it should still prevent the warning from being raised even if the user doesn't configure the LLM report provider correctly.
- The `rootpath` and `stash` options are not affected by the LLM report provider configuration and can be set to any value without triggering the warning.
- Setting `llm_aggregate_dir`, `llm_aggregate_policy`, `llm_aggregate_run_id`, `llm_aggregate_group_id` to None should prevent the warning from being raised if they are not configured correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	52 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-285, 287, 289, 291, 298, 305-307, 310, 319, 321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343-344, 347, 349)
src/pytest_llm_report/plugin.py	29 lines (ranges: 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236-

	238, 240-241, 245-246, 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_validation_errors` 3ms 3

AI ASSESSMENT

Scenario: Test that validation errors raise UsageError when invalid configuration is provided.

Why Needed: To prevent a potential bug where the plugin does not handle invalid configuration properly and raises a UsageError.

Key Assertions:

- mock_config.option.llm_report_html is None
- mock_config.option.llm_report_json is None
- mock_config.option.llm_report_pdf is None
- mock_config.option.llm_evidence_bundle is None
- mock_config.option.llm_dependency_snapshot is None
- mock_config.option.llm_requests_per_minute is None
- mock_config.option.llm_aggregate_dir is None
- mock_config.option.llm_aggregate_policy is None
- mock_config.option.llm_aggregate_run_id is None
- mock_config.option.llm_aggregate_group_id is None
- mock_config.option.llm_max_retries is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	51 lines (ranges: 123, 163, 191, 194-197, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 276, 279-281, 283, 285, 287, 289, 291, 298, 305-307, 310, 319, 321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343-344, 347, 349)
src/pytest_llm_report/plugin.py	25 lines (ranges: 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-232, 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_worker_skip` 1ms 2

AI ASSESSMENT

Scenario: Test that configure skips on xdist workers and verifies the correct behavior.

Why Needed: This test prevents a potential regression where the plugin might not skip configuration on xdist workers, potentially leading to incorrect results or skipped tests.

Key Assertions:

- The `pytest_configure` function should be called early without calling `addinivalue_line`.
- The `addinivalue_line` method should still be called for markers before the worker check.
- The plugin should skip configuration on xdist workers 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`	17 lines (ranges: 202-204, 206-208, 210-212, 216-217, 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigureFallback::test_pytest_configure_fallback_load` 3ms 2

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	29 lines (ranges: 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236-238, 240-241, 245-246, 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_all_ini_options` 2ms 3

AI ASSESSMENT

Scenario: Test loading all INI options for plugin load configuration.

Why Needed: Prevents regression in plugin load configuration when CLI options are not set.

Key Assertions:

- The provider is set to 'ollama' as expected.
- The model is set to 'llama3.2' as expected.
- The context mode is set to 'complete' as expected.
- The number of requests per minute is set to 10 as expected.
- The report HTML is set to 'ini.html' as expected.
- The report JSON is set to 'ini.json' as expected.
- The directory for aggregate files is set to '/project' as expected.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	36 lines (ranges: 123, 163, 276, 279-291, 298, 305-307, 310, 319, 321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343-344, 347, 349)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_cli_overrides_ini` 2ms 3

AI ASSESSMENT

Scenario: Test CLI options override INI options.

Why Needed: This test prevents a bug where the plugin's report configuration is not correctly overridden from INI settings to command-line arguments.

Key Assertions:

- The 'report_html' option in the config should be set to 'cli.html'.
- The 'report_json' option in the config should be set to 'cli.json'.
- The 'report_pdf' option in the config should be set to 'cli.pdf'.
- The 'report_evidence_bundle' option in the config should be set to 'bundle.zip'.
- The 'report_dependency_snapshot' option in the config should be set to 'deps.json'.
- The 'llm_requests_per_minute' option in the config should be set to 20.
- The 'aggregate_dir' option in the config should be set to '/agg'.
- The 'aggregate_policy' option in the config should be set to 'merge'.
- The 'aggregate_run_id' option in the config should be set to 'run-123'.
- The 'aggregate_group_id' option in the config should be set to 'group-abc'.
- The rootpath of the config should be set to '/project'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	43 lines (ranges: 123, 163, 276, 279, 281, 283, 285, 287-291, 298, 305-307, 310, 319-332, 335-344, 347, 349)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::test_terminal_summary_disabled` 1ms 2

AI ASSESSMENT

Scenario: Test that terminal summary skips when plugin is disabled.

Why Needed: Prevents a regression where the plugin's terminal summary is not properly handled when it is disabled.

Key Assertions:

- The stash.get method of the config object was called with the correct key and value.
- The stash.get method of the config object was called once.
- The stash.get method of the config object returned False for enabled.
- The stash.get method of the config object did not return None when the plugin is disabled.
- The stash.get method of the config object correctly checked if the plugin is enabled before returning 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	9 lines (ranges: 271, 275-276, 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::test_terminal_summary_worker_skip` 1ms 2

AI ASSESSMENT

Scenario: Test that terminal summary skips on xdist worker when configured correctly.

Why Needed: This test prevents a regression where the plugin fails to skip terminal summaries on xdist workers due to incorrect configuration.

Key Assertions:

- The `pytest_terminal_summary` function should return early without doing anything when the `workerinput` is set to a valid worker ID.
- The `workerinput` parameter of `pytest_terminal_summary` should be a dictionary with a single key-value pair where the key is 'workerid' and the value is a string representing the worker ID.
- The `result` variable of `pytest_terminal_summary` should be `None` when called with the provided arguments.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	8 lines (ranges: 271-272, 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::testload_config` 3ms 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`	41 lines (ranges: 123, 163, 276, 279, 281, 283, 285, 287, 289, 291, 298, 305-307, 310, 319-332, 335-344, 347, 349)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_disabled` 2ms 2

AI ASSESSMENT

Scenario: Test makereport skips when disabled.

Why Needed: The test prevents a regression where the plugin does not report any issues even though makereport is disabled.

Key Assertions:

- mock_item.config.stash.get() returns False
- mock_call.send() raises StopIteration exception
- mock_outcome.get_result().get_result() 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/plugin.py	7 lines (ranges: 413-414, 417-418, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_enabled` 2ms 2

AI ASSESSMENT

Scenario: Test makereport calls collector when enabled.

Why Needed: Prevents a potential bug where the collector is not called when makereport is enabled.

Key Assertions:

- The `pytest_runtest_makereport` function should be called with the `mock_collector` object as its `collector_key` argument.
- The `mock_collector` object should have a `handle_runtest_logreport` method that calls the `makereport` function.
- The `make_report` function should return an instance of `MockReport` or similar.
- The `collectors` dictionary in `pytest_llm_report` plugin should contain the `mock_collector` object as one of its values.
- The `pytest_runtest_makereport` function should be able to handle a mock collector without raising an exception.
- The `pytest_runtest_makereport` function should not raise a StopIteration exception when called with a mock outcome.
- The `mock_collector.handle_runtest_logreport` method should call the `make_report` function and return a result object.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_disabled` 1ms 2

AI ASSESSMENT

Scenario: Test that collection_finish is skipped when the plugin's collection finish flag is disabled.

Why Needed: The test prevents a regression where collection_finish is not executed correctly when the plugin's collection finish flag is disabled.

Key Assertions:

- mock_session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_collection_finish(mock_session) should be called with _enabled_key set to False
- mock_session.config.stash.get.return_value should be False
- _enabled_key should not be set in the mock session config
- collection_finish should skip the collection process when collection finish flag is disabled

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 413-414, 417, 421-423, 457-458)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_enabled` 2ms 2

AI ASSESSMENT

Scenario: Test that Pytest collection finish is called when collection_finish is enabled.

Why Needed: This test prevents a potential regression where the collector does not call stash_get for certain keys.

Key Assertions:

- The pytest_collection_finish function should be called with mock_session.items as argument.
- The stash_get function should return True for _enabled_key and False for _collector_key.
- The stash_get function should return mock_collector for _enabled_key and None for _collector_key.
- mock_collector.handle_collection_finish should be called once with mock_session.items as argument.
- mock_collector.handle_collection_finish should not be called twice (once before and once after pytest_collection_finish is called).
- The stash_get function should return the correct default value in all cases.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	10 lines (ranges: 413-414, 417, 421-423, 457, 461-463)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_disabled` 1ms 2

AI ASSESSMENT

Scenario: Test that sessionstart skips when disabled and verifies the expected behavior.

Why Needed: This test prevents a potential regression where pytest_sessionstart does not check for enabled status correctly when sessionstart is disabled.

Key Assertions:

- The `pytest_sessionstart` function should have been called with the `_enabled_key` key set to `False` before checking the stash.
- The `get` method of the `stash` attribute of the mock session object should have returned `False` instead of a value indicating enabled status.
- The `assert_called_with` method on the `get` call of the mock session object should not have been called with any arguments.
- The stash attribute of the mock session object should still be set to its original value (i.e., `True`) after the test has finished.
- The `_enabled_key` key in the mock session object's configuration should still be present and valid.
- The `pytest_sessionstart` function should not have been called with any arguments or keys when sessionstart is disabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	8 lines (ranges: 413-414, 417, 421-423, 474-475)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_enabled` 1ms 3

AI ASSESSMENT

Scenario: Test that sessionstart initializes collector when enabled and creates a stash with the necessary keys.

Why Needed: This test prevents a potential bug where the collector is not created or does not have access to the necessary configuration data, leading to incorrect reporting of Pytest sessions.

Key Assertions:

- assert _collector_key in mock_stash
- assert _start_time_key in mock_stash

COVERAGE

src/pytest_llm_report/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: 413-414, 417, 421-423, 474, 478, 481, 483-484)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption` 2ms 2

AI ASSESSMENT

Scenario: Verify pytest_addoption adds expected arguments to the parser.

Why Needed: pytest_addoption may not add all required arguments if the 'llm-report' group is not properly configured.

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	124 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption_ini` 2ms 2

AI ASSESSMENT

Scenario: Test pytest_addoption adds INI options for plugin terminal summary.

Why Needed: This test prevents a regression where the plugin does not add INI options to the command line when using pytest.

Key Assertions:

- Verify that 'llm_report_html' is added to the ini list.
- Verify that 'llm_report_json' is added to the ini list.
- Verify that 'llm_report_max_retries' is added to the ini list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	124 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_coverage_calculation` 4ms 3

AI ASSESSMENT

Scenario: Test coverage percentage calculation logic for terminal summary.

Why Needed: Prevents regression in coverage reporting when terminal summary is enabled and a coverage file exists.

Key Assertions:

- The `mock_cov.load` method should be called with the correct arguments (None, mock_config).
- The `mock_cov.report` method should be called with the correct arguments (None, mock_config).
- The `pytest_terminal_summary` function should not raise an exception when coverage file does not exist.
- The `coverage.CoverageMapper` class is correctly instantiated and configured.
- The `Coverage` instance returned by `mock_cov.report` has a valid percentage value.
- The `ReportWriter` instance created with the mock configuration has been properly initialized.
- No exceptions are raised when calling `pytest_terminal_summary` with an invalid configuration.

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: 271, 275, 279, 282, 301-302, 304, 306, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-338, 340, 342-345, 357-358, 363-364, 391-401, 413-414, 417, 421-423)

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 annotations correctly and reports the correct provider.

Why Needed: This test prevents regression in reporting when LLM is enabled.

Key Assertions:

- The `pytest_terminal_summary_llm_enabled` function should report the correct provider (in this case, 'ollama') even when LLM is enabled.
- The `pytest_terminal_summary_llm_enabled` function should pass the provided configuration to the annotation process.
- The annotation process should verify that the correct model name ('gpt-4') was used in the terminal summary report.
- The `pytest_terminal_summary_llm_enabled` function should not raise any exceptions when LLM is enabled.

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: 271, 275, 279, 282, 301-302, 304, 306, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357-358, 363-366, 369, 371, 374-376, 383-388, 391-401, 413-414, 417, 421-423)

PASSED tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_no_collector 2ms 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	45 lines (ranges: 271, 275, 279, 282, 301-302, 304, 306, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-364, 391-401, 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_with_aggregation` 2ms 3

AI ASSESSMENT

Scenario: Test terminal summary with aggregation enabled.

Why Needed: This test prevents a regression where the plugin does not aggregate terminal summaries correctly when reporting to JSON.

Key Assertions:

- The `aggregate_dir` is set to `/agg` in the configuration.
- The aggregator function is called once and returns a report.
- The report writer writes JSON and HTML files.
- The aggregation result is stored in the stash.
- The stash supports both get() and [] indexing.
- The terminal summary is written to the correct directory.
- The terminal summary contains the aggregated data.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	21 lines (ranges: 271, 275, 279, 282-283, 285-286, 289-290, 292, 294-298, 413-414, 417, 421-423)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummaryErrors::test_terminal_summary_coverage_error` 4ms 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: 271, 275, 279, 282, 301-302, 304, 306, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-338, 348-351, 357-358, 363-364, 391-401, 413-414, 417, 421-423)

📄 tests/test_prompts.py

6 tests

PASSED `tests/test_prompts.py::TestContextAssembler::test_assemble_balanced_context` 7ms 4

AI ASSESSMENT

Scenario: Tests the ContextAssembler to assemble a balanced context for a test file.

Why Needed: This test prevents regression when assembling contexts with unbalanced llm_context_mode, as it ensures that all required dependencies are included.

Key Assertions:

- The 'utils.py' file is present in the assembled context.
- The 'def util()' function is found in the 'utils.py' file within the assembled context.
- All lines of the 'utils.py' file are covered by the test.
- The line ranges and line count of the coverage entry match the expected values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/prompts.py	51 lines (ranges: 33, 49, 52, 55, 58, 60-61, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 116, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-155, 158-159, 163, 191-192, 194)

PASSED `tests/test_prompts.py::TestContextAssembler::test_assemble_complete_context` 1ms 4

AI ASSESSMENT

Scenario: Assembling a complete context for the 'test_a.py' test file.

Why Needed: This test prevents regression when the 'test_a.py' test file is modified to include new tests or changes to existing code.

Key Assertions:

- The source of the assembled context should contain the function `test_1` from the 'test_a.py' test file.
- The assembled context should have a node representing the function `test_1` in the correct location.
- The assembled context should not include any other functions or nodes from the 'test_a.py' test file.
- The source of the assembled context should be identical to the original 'test_a.py' test file.
- The assembled context should have the same node structure as the original 'test_a.py' test file.
- Any changes made to the 'test_a.py' test file should not affect the assembled context.
- The assembled context should still contain the correct function `test_1` even if it is moved or renamed in the 'test_a.py' test file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 that the ContextAssembler can assemble a minimal context for a test file with a single test function.

Why Needed: This test prevents regression when using the 'minimal' llm_context_mode, as it ensures that only the necessary code is assembled into the context.

Key Assertions:

- The source of the test function should contain the function being tested.
- The context of the test file should be empty.
- The assembler should assemble a minimal context for the test file.
- The assembler should not assemble any unnecessary code into the context.
- The assembler should return an empty context if no tests are found in the test file.
- The assembler should raise an error if the 'minimal' llm_context_mode is not supported.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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

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: 413-414, 417, 421-423)
src/pytest_llm_report/prompts.py	34 lines (ranges: 33, 49, 52, 55, 58, 60-61, 65, 78-79, 82-84, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-156, 158-159, 163, 191-192, 194)

PASSED `tests/test_prompts.py::TestContextAssembler::test_get_test_source_edge_cases` 1ms 4

AI ASSESSMENT

Scenario: Verify the correct handling of non-existent files and nested test names with parameters.

Why Needed: This test prevents a potential bug where the ContextAssembler does not correctly handle cases where it encounters an unknown file or nested test name with parameters.

Key Assertions:

- The function `_get_test_source` returns an empty string when given a non-existent file.
- The function `_get_test_source` correctly identifies and includes the entire nested test name in its output, including any 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: 413-414, 417, 421-423)
src/pytest_llm_report/prompts.py	26 lines (ranges: 33, 78-79, 82-84, 86-87, 92, 94-95, 98-101, 103-112, 116)

PASSED `tests/test_prompts.py::TestContextAssembler::test_should_exclude` 1ms 4

AI ASSESSMENT

Scenario: The test verifies that the ContextAssembler should exclude certain files from being processed by LLM.

Why Needed: This test prevents a potential bug where the ContextAssembler incorrectly excludes important files, leading to unexpected behavior or errors in the LLM's output.

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: 413-414, 417, 421-423)
src/pytest_llm_report/prompts.py	5 lines (ranges: 33, 191-194)

📄 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 when consecutive lines are compressed to include all elements within a range.

Key Assertions:

- The input list is not empty.
- All elements in the list are present in the output.
- The start value of the range is correct (i.e., it's less than or equal to the end value).
- The end value of the range is correct (i.e., it's greater than or equal to the start value).
- All elements within the range are present in the output.
- The total number of elements in the input list is preserved in the output.
- No duplicate ranges are created in the output.
- The order of consecutive lines is maintained in the output.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 `compress_ranges` should correctly identify and return the compressed range for a list containing duplicate values.

Why Needed: This test prevents bugs that may occur when compressing ranges with duplicates, such as incorrect or missing ranges.

Key Assertions:

- asserts that the output of `compress_ranges([1, 2, 2, 3, 3, 3])` is '1-3'
- asserts that the function correctly handles duplicate values by returning a single range ('1-3')
- asserts that the function returns an empty string for an input list with 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/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

PASSED tests/test_ranges.py::TestCompressRanges::test_empty_list 1ms 3

AI ASSESSMENT

Scenario: Testing the `compress_ranges` function with an empty input.

Why Needed: This test prevents a potential bug where the function returns an incorrect result for an empty input list.

Key Assertions:

- The function should return an empty string when given an empty list as input.
- The function should not raise any exceptions or errors when given an empty list as input.
- The function should correctly handle and return an empty string for the compressed range.
- The function should not produce unexpected results or incorrect values for the compressed range of an empty list.
- The function should be able to handle large inputs, including empty lists, 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/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/ranges.py	2 lines (ranges: 29-30)

PASSED `tests/test_ranges.py::TestCompressRanges::test_mixed_ranges` 1ms 3

AI ASSESSMENT

Scenario: Test compressing mixed ranges.

Why Needed: Prevents regression in cases where the input contains a mix of single values and range values.

Key Assertions:

- The output should be '1-3, 5, 10-12, 15'.
- The output should contain all the given ranges.
- The output should not contain any invalid or missing ranges.
- The output should have a consistent order and no duplicates.
- All numbers in the input list should be present in the output.
- No single value should be out of range (i.e., less than 1 or greater than 15).
- No range should start before its end (e.g., '1-2' but not '2-1').

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`
`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

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: 413-414, 417, 421-423)`
`src/pytest_llm_report/util/ranges.py`	`14 lines (ranges: 29, 33, 35-37, 39-40, 45-47, 50, 52, 65-66)`

PASSED tests/test_ranges.py::TestCompressRanges::test_single_line 1ms 3

AI ASSESSMENT

Scenario: The 'single_line' test verifies that a single-line input does not utilize the range notation.

Why Needed: This test prevents regression where the function compresses only one element in a list.

Key Assertions:

- asserts that the output of `compress_ranges([5])` is equal to '5'.
- checks if the compressed value matches the expected result.
- verifies that the input list has only one element.
- ensures that the function does not use range notation for a single-element list.
- tests the case where the input list contains multiple elements, but the function still compresses it correctly.
- checks if the compressed value is correct even when the input list contains duplicate values.
- verifies that the function handles empty lists correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 function `compress_ranges` is expected to correctly handle two consecutive line numbers when compressing ranges.

Why Needed: This test prevents a potential bug where the function incorrectly handles consecutive line numbers as separate ranges.

Key Assertions:

- assert '1-2' in str(compress_ranges([1, 2]))
- assert '1,2' not in str(compress_ranges([1, 2]))
- assert '2-3' not in str(compress_ranges([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: 413-414, 417, 421-423)
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: The test verifies that the `compress_ranges` function correctly handles an unsorted list of integers.

Why Needed: This test prevents a potential bug where the function would incorrectly group numbers in ascending order instead of their actual ranges.

Key Assertions:

- asserts that the output is correct for input `[5, 1, 3, 2]`
- the output should be `1-3, 5`
- the list elements are sorted in ascending order
- the function groups numbers by their actual ranges 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: 413-414, 417, 421-423)
src/pytest_llm_report/util/ranges.py	16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)

PASSED `tests/test_ranges.py::TestExpandRanges::test_empty_string` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that an empty string expands to an empty list.

Why Needed: This test prevents a potential bug where an empty string is not expanded correctly.

Key Assertions:

- assert expand_ranges('') == []
- assert expand_ranges([]) == []
- assert expand_ranges(['']) == []

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 a mixed set of ranges and singles.

Why Needed: This test prevents regression in cases where the input contains both range notation (e.g., '1-3') and non-range values (e.g., '10').

Key Assertions:

- The function should split the comma-separated ranges into individual numbers correctly.
- It should handle single values without splitting them.
- It should ignore invalid or missing ranges.
- The resulting list should contain all specified numbers in the correct order.
- It should preserve the original order of non-range values (e.g., '5' and '10-12').
- It should raise an error for invalid input (e.g., '1, 3, abc') to prevent unexpected behavior.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 expand a range of numbers.

Why Needed: This test prevents the function from expanding the range incorrectly, potentially leading to incorrect results or errors.

Key Assertions:

- The input string should be in the format 'start-end' (e.g., '1-3')
- The start value should be less than or equal to the end value
- The function should return a list of numbers between the start and end values

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/ranges.py	10 lines (ranges: 81, 84-91, 95)

PASSED tests/test_ranges.py::TestExpandRanges::test_roundtrip 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `compress_ranges` and `expand_ranges` functions produce the same output when given a list of numbers.

Why Needed: This test prevents regression in cases where the input list is modified or reordered during compression or expansion.

Key Assertions:

- The compressed list should be identical to the original list.
- The expanded list should contain all elements from the original list.
- No duplicate elements should be present in the expanded list.
- All numbers in the original list should be present in the expanded list.
- The order of elements in the original and expanded lists should be preserved.
- The compressed list should not have any additional elements compared to the original list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 function `expand_ranges` is expected to handle a single input '5' correctly.

Why Needed: This test prevents a potential bug where the function would incorrectly return a list with multiple elements for a single number.

Key Assertions:

- the input string '5' should be converted into a list containing only one element, which is 5 itself.
- the expected output of `expand_ranges('5')` should be `[5]` exactly.
- any other input strings with multiple elements should not produce lists with more than one element.
- for inputs like 'abc', the function should return an empty list or raise a meaningful error.
- for inputs like '1,2,3', the function should return a list containing only 1, 2 and 3.
- the function should handle edge cases where the input string is empty.
- any non-numeric characters in the input string should be ignored.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/ranges.py	7 lines (ranges: 81, 84-87, 93, 95)

📄 tests/test_render.py

9 tests

PASSED `tests/test_render.py::TestFormatDuration::test_milliseconds` 1ms 3

AI ASSESSMENT

Scenario: Test the formatting of duration values less than 1 second.

Why Needed: Prevents regression in handling durations under 1s, where the expected output may not be accurate.

Key Assertions:

- The function `format_duration(0.5)` should return '500ms'.
- The function `format_duration(0.001)` should return '1ms'.
- The function `format_duration(0.0)` should return '0ms'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/render.py	2 lines (ranges: 65, 67)

PASSED `tests/test_render.py::TestFormatDuration::test_seconds` 1ms 3

AI ASSESSMENT

Scenario: Test that the function formats duration correctly for seconds.

Why Needed: Prevents a potential bug where the function does not handle durations greater than or equal to 1 second correctly.

Key Assertions:

- {'message': "Expected format_duration(1.23) to return '1.23s'"}
- {'message': "Expected format_duration(60.0) to return '60.00s'"}
- {'message': 'Function should handle durations greater than or equal to 1 second correctly'}

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 when the outcome is 'xfailed' as it would incorrectly map it to 'outcome-xfailed'.

Key Assertions:

- outcome_to_css_class('passed') == 'outcome-passed'
- outcome_to_css_class('failed') == 'outcome-failed'
- outcome_to_css_class('skipped') == 'outcome-skipped'
- outcome_to_css_class('xfailed') == 'outcome-xfailed'
- outcome_to_css_class('xpassed') == 'outcome-xpassed'
- outcome_to_css_class('error') == 'outcome-error'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/render.py	8 lines (ranges: 79-85, 87)

PASSED tests/test_render.py::TestOutcomeToCssClass::test_unknown_outcome 1ms 3

AI ASSESSMENT

Scenario: tests/test_render.py::TestOutcomeToCssClass::test_unknown_outcome

Why Needed: This test prevents a regression where the `outcome_to_css_class` function returns an incorrect default class for unknown outcomes.

Key Assertions:

- The `outcome_to_css_class` function should return 'outcome-unknown' when given an unknown outcome.
- The `outcome_to_css_class` function should not raise an exception when given an unknown outcome.
- The `outcome_to_css_class` function should correctly handle the case where the outcome is not recognized.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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

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: 413-414, 417, 421-423)
src/pytest_llm_report/render.py	52 lines (ranges: 65-67, 79-85, 87, 121-124, 126-127, 131-132, 141-143, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED tests/test_render.py::TestRenderFallbackHtml::test_renders_coverage 1ms 3

AI ASSESSMENT

Scenario: Test verifies that the test renders coverage information.

Why Needed: This test prevents a regression where the coverage report is missing or incomplete.

Key Assertions:

- The 'src/foo.py' file should be included in the rendered HTML.
- The number of lines covered should be reported correctly (in this case, 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: 413-414, 417, 421-423)
src/pytest_llm_report/render.py	52 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-129, 131-132, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_llm_annotation` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the LLM annotation is included in the rendered HTML report.

Why Needed: This test prevents a potential security vulnerability where an attacker could bypass authentication by manipulating the HTML content of the report.

Key Assertions:

- LLMAnnotation contains the string 'Tests login flow' in its text.
- LLMAnnotation contains the string 'Prevents auth bypass' in its text.
- The LLMAnnotation is present in the rendered HTML report.
- The presence of 'Tests login flow' and 'Prevents auth bypass' in the LLMAnnotation's text indicates that authentication was successful.
- The inclusion of 'Tests login flow' and 'Prevents auth bypass' in the LLMAnnotation suggests that the test ran without any issues.
- The rendered HTML report contains both 'Tests login flow' and 'Prevents auth bypass' as expected.
- The presence of these strings in the LLMAnnotation indicates a successful authentication process.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/render.py	54 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-127, 131-134, 136-137, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_source_coverage` 1ms 3

AI ASSESSMENT

Scenario: Test renders source coverage to ensure it is included in the HTML output.

Why Needed: This test prevents a potential regression where the source coverage summary might not be displayed correctly if there are missing or unreported tests.

Key Assertions:

- The 'Source Coverage' section should be present in the rendered HTML.
- The source code path 'src/foo.py' should be included in the HTML output.
- The percentage of covered statements should be displayed as '80.0%' in the HTML output.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 413-414, 417, 421-423)`
`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: Tests the rendering of xfailed and xpassed summary entries.

Why Needed: This test prevents regression where a user submits multiple tests with different outcomes (xfailed and xpass) without seeing their respective counts in the summary.

Key Assertions:

- The 'XFailed' string is present in the rendered HTML.
- The 'XPassed' string is present in the rendered HTML.
- Both 'XFailed' and 'XPassed' strings are included in the rendered HTML.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 output (different content) and potentially cause issues with reporting or data integrity.

Key Assertions:

- hash1 should be different from hash2
- hash1 is not equal to hash2 using == operator
- hash1 is not equal to hash2 using != operator
- hash1 is not equal to hash2 using != operator (Python's built-in comparison)
- hash1 is not equal to hash2 when comparing bytes objects directly
- hash1 and hash2 should have different hexadecimal 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/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

PASSED tests/test_report_writer.py::TestComputeSha256::test_empty_bytes 1ms 3

AI ASSESSMENT

Scenario: Test 'Empty bytes should produce consistent hash' verifies that an empty byte string produces a hash with the same value and correct length.

Why Needed: This test prevents a potential bug where an empty byte string is expected to produce a different or incorrect hash.

Key Assertions:

- The two computed hashes should be equal (i.e., they should have the same value).
- The length of each computed hash should be exactly 64 characters (i.e., it should match the SHA256 hex length).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	1 lines (ranges: 55)

PASSED tests/test_report_writer.py::TestReportWriter::test_build_run_meta 5ms 4

AI ASSESSMENT

Scenario: Test that the build_run_meta method includes version info and other expected metadata.

Why Needed: This test prevents regression where the ReportWriter does not include version information in the run meta.

Key Assertions:

- The duration of the run should be 60 seconds.
- The pytest version should have a value.
- The plugin version should be '0.1.0'.
- The python version should also match the one used to create the ReportWriter 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/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 potential regression where the summary does not include all outcome types, leading to incorrect reporting.

Key Assertions:

- total == 6
- passed == 1
- failed == 1
- skipped == 1
- xfailed == 1
- xpassed == 1
- error == 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: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	19 lines (ranges: 156-158, 312, 314-315, 317-328, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_build_summary_counts` 1ms 4

AI ASSESSMENT

Scenario: The test verifies that the `build_summary` method correctly counts outcomes in a report.

Why Needed: This test prevents a bug where the total count of outcomes is not accurate due to incorrect handling of skipped tests.

Key Assertions:

- assert summary.total == 4
- assert summary.passed == 2
- assert summary.failed == 1
- assert summary.skipped == 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: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	13 lines (ranges: 156-158, 312, 314-315, 317-322, 330)

PASSED tests/test_report_writer.py::TestReportWriter::test_create_writer 1ms 4

AI ASSESSMENT

Scenario: Test that a new ReportWriter instance is created with the provided configuration.

Why Needed: This test prevents a potential bug where the writer's configuration and warnings/artifacts are 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 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: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	3 lines (ranges: 156-158)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_assembles_tests` 5ms 4

AI ASSESSMENT

Scenario: Test writes a report that includes all tests, but does not handle cases where output paths are specified.

Why Needed: This test prevents regression by ensuring that the report writer can write reports even if no output paths are provided.

Key Assertions:

- The number of tests in the report should be equal to 2.
- The total number of tests in the summary should also be 2.
- All tests should have been written to the specified output path(s).

COVERAGE

src/pytest_llm_report/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: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	93 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_includes_coverage_percent` 6ms 4

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class writes a report with a total coverage percentage that matches the provided value.

Why Needed: This test prevents a regression where the coverage percentage is not accurately reported in reports.

Key Assertions:

- The `coverage_total_percent` attribute of the report summary should match the provided `coverage_percent` value.
- The `report.write_report()` method should return an instance with a `summary` attribute that has a `coverage_total_percent` attribute matching the provided value.
- The `report.summary.coverage_total_percent` attribute should be equal to the provided `coverage_percent` value after writing the report.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	93 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-199, 202-206, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_includes_source_coverage` 5ms 4

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_includes_source_coverage verifies that the report includes source coverage summary.

Why Needed: This test prevents a regression where the report does not include source coverage information, potentially misleading users about the code's quality.

Key Assertions:

- The length of `report.source_coverage` is 1.
- The file path of `report.source_coverage[0]` is `src/foo.py`.
- All statements in `source_coverage` are covered by the report.
- At least one statement in `source_coverage` is missed by the report.
- The coverage percentage of `source_coverage` is 87.5%.
- The ranges where code is missed (`missed_ranges`) are `src/foo.py:5`.
- All statements in `source_coverage` are covered by the report.
- At least one statement in `source_coverage` is missed by the report.
- The coverage percentage of `source_coverage` is 87.5%.
- The ranges where code is missed (`missed_ranges`) are `src/foo.py:1-4, src/foo.py:6-7`.

COVERAGE

src/pytest_llm_report/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: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	92 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_merges_coverage` 5ms 4

AI ASSESSMENT

Scenario: Test Report Writer: test_write_report_merges_coverage verifies that the report writer merges coverage into tests.

Why Needed: This test prevents regression where a merge of coverage does not correctly update the test results.

Key Assertions:

- The number of coverage lines in the first test should be 1.
- The file path of the first coverage line should match 'src/foo.py'.
- All other tests' coverage lines should have a file path that matches their test 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	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	94 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186-189, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_atomic_write_fallback` 6ms 5

AI ASSESSMENT

Scenario: Test that the report writer falls back to direct write if atomic write fails and the 'os.replace' mock is used.

Why Needed: This test prevents a regression where an atomic write operation fails, causing the report writer to fall back to direct write instead of using the atomic write mechanism.

Key Assertions:

- The file `report.json` should exist at the specified path.
- Any warnings with code 'W203' should be present in the `writer.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	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: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	125 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506-507, 509-512, 515-516)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_creates_directory_if_missing` 6ms 5

AI ASSESSMENT

Scenario: Test verifies that the `ReportWriter` creates an output directory if it doesn't exist.

Why Needed: This test prevents a regression where the report writer fails to create the output directory when it is missing.

Key Assertions:

- The output directory should be created at the specified path.
- The file `report.json` should be present in the output directory.
- If the output directory does not exist, the `ReportWriter` should attempt to create it.
- If the report is written successfully, the output directory should contain the expected files.
- The test should fail if the output directory already exists and contains the expected files.
- The test should pass if the output directory is created correctly and the file `report.json` is 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/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: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	123 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-477, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_ensure_dir_failure` 1ms 4

AI ASSESSMENT

Scenario: Test ensures that directory creation fails and reports warnings when permission is denied.

Why Needed: This test prevents a potential bug where the report writer does not handle directory creation failures correctly, potentially leading to silent errors or unexpected behavior.

Key Assertions:

- The `writer._ensure_dir(json_path)` method should raise an OSError with code 'W201' when the directory creation fails.
- Any warnings raised by the `writer.warnings` list should have a code of 'W201'.
- The test should fail if no warnings are raised in the case where the directory creation 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/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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` 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/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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` 34ms 6

AI ASSESSMENT

Scenario: Test verifies that the report writer creates an HTML file.

Why Needed: This test prevents a regression where the report writer fails to create an HTML file for reports with multiple tests.

Key Assertions:

- The report.html file should exist.
- The report.html file should contain the expected content.
- The report.html file should include all the nodes from the tests.
- The report.html file should display 'test1' and 'test2' in the 'Passed' section.
- The report.html file should display 'FAILED' and 'Skipped' in the 'Failed' section.
- The report.html file should display 'XFailed' and 'XPassed' in the 'Errors' section.
- The report.html file should contain a header with the expected content.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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` 34ms 6

AI ASSESSMENT

Scenario: The test verifies that the report writer includes xfail outcomes in the HTML summary.

Why Needed: This test prevents a regression where the xfail summary is not included in the report.

Key Assertions:

- The 'XFAILED' string should be present in the HTML summary.
- The 'XFailed' string should also be present in the HTML summary.
- The 'XPASSED' string should be present in the HTML summary.
- The 'XPassed' string should also be present in the HTML summary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	118 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321, 323-326, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_json_creates_file` 6ms 5

AI ASSESSMENT

Scenario: Test verifies that a JSON file is created with the report.

Why Needed: This test prevents regression where the report writer does not create a JSON file.

Key Assertions:

- The JSON file should be created at the specified path.
- The report should have an artifact tracked.
- At least one artifact should be present in the artifacts list.
- The number of artifacts should be greater than or equal to 1.
- The JSON file should exist at the specified path.
- The ReportWriter instance should successfully write a report with the given 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	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: 413-414, 417, 421-423)
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` 38ms 6

AI ASSESSMENT

Scenario: Test verifies that the `write_pdf` method creates a PDF file when Playwright is available.

Why Needed: This test prevents regression where the `write_pdf` method does not create a PDF file even if Playwright is available.

Key Assertions:

- The `write_pdf` method should write the contents of `page.pdf` to `pdf_path`.
- Any artifacts created by the report writer should have a path that matches `pdf_path`.
- All artifacts should be present in the `writer.artifacts` list.
- If `report_pdf` is set to a file path, it should be created as a PDF file.
- The `write_pdf` method should not raise an exception if Playwright is not available.
- The `write_pdf` method should use the correct file extension for the PDF file (`.pdf`).
- If `report_pdf` is set to a relative path, it should be resolved correctly to the desired location.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 413-414, 417, 421-423)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	125 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230-231, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 401, 410, 412, 414-423, 434-435, 437-443, 448, 453, 455, 458-462, 470-471)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_pdf_missing_playwright_warns` 6ms 4

AI ASSESSMENT

Scenario: Test should warn when Playwright is missing for PDF output.

Why Needed: To prevent a silent failure where the report is generated without any warnings or errors.

Key Assertions:

- The file 'report.pdf' should not exist.
- Any warning with code W204_PDF_PLAYWRIGHT_MISSING should be present in the list of warnings.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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_v2.py

2 tests

PASSED `tests/test_report_writer_coverage_v2.py::test_report_writer_ensure_dir_creation` 1ms 4

AI ASSESSMENT

Scenario: Verifies that the report writer creates a directory with an error message when it already exists.

Why Needed: Prevents a potential bug where the report writer does not create a new directory even if the existing one is empty.

Key Assertions:

- The `tmp_dir` directory should exist before creating any warnings in the report.
- Any warning code should be 'W202'.
- The `writer.warnings` list should contain at least one warning with code 'W202'.

COVERAGE

src/pytest_llm_report/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: 413-414, 417, 421-423)
src/pytest_llm_report/report_writer.py	11 lines (ranges: 156-158, 470-477)

PASSED `tests/test_report_writer_coverage_v2.py::test_report_writer_metadata_skips` 10ms 5

AI ASSESSMENT

Scenario: Verify that the report_writer_metadata_skips test prevents a bug where metadata is skipped when reports are disabled.

Why Needed: This test ensures that the report_writer_metadata_skips function behaves correctly when reports are disabled, preventing metadata from being skipped.

Key Assertions:

- The 'start_time' key should be present in the metadata dictionary.
- The 'llm_model' key should not be present in the metadata dictionary when reports are disabled.
- The report_writer_metadata_skips function should raise an AssertionError with a meaningful error message when reports are disabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	36 lines (ranges: 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425)
src/pytest_llm_report/options.py	2 lines (ranges: 123, 163)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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` creates a valid annotation from a dictionary with all required fields.

Why Needed: Prevents regression in cases where the input data does not contain all necessary fields for an annotation.

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: 413-414, 417, 421-423)

PASSED tests/test_schemas.py::TestAnnotationSchema::test_to_dict_full 1ms 3

AI ASSESSMENT

Scenario: Should convert to dictionary with all fields.

Why Needed: Prevent regression in authentication logic.

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: 413-414, 417, 421-423)

📄 tests/test_smoke_pytester.py

15 tests

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_report_created` 88ms 8

AI ASSESSMENT

Scenario: Test that an HTML report is created when running the test with --llm-report.

Why Needed: This test prevents a regression where the report generation fails without providing any error message.

Key Assertions:

- The file "report.html" exists in the specified path.
- The content of the "report.html" file contains the string '
- The content of the "report.html" file contains the string 'test_simple', which is also expected to be present.

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	58 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319-321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	191 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332,

	335-336, 357, 363-364, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	101 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_summary_counts_all_statuses` 125ms 8

AI ASSESSMENT

Scenario: test_html_summary_counts_all_statuses verifies that HTML summary counts include all statuses.

Why Needed: This test prevents regression where the report does not include all statuses, such as when there are no tests or only failed tests.

Key Assertions:

- The 'Total Tests' label should be included in the summary.
- The 'Passed' label should have a count of 1.
- The 'Failed' label should have a count of 1.
- The 'Skipped' label should have a count of 1.
- The 'XFailed' and 'XPassed' labels should both have counts of 1.
- The 'Errors' and 'Error' labels should both have counts of 1.

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	58 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319-321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	191 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184,

	186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-364, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
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` 75ms 7

AI ASSESSMENT

Scenario: The JSON report is created and its existence is verified.

Why Needed: This test prevents a regression where the pytester does not create a JSON report even when tests are run with --llm-report-json flag.

Key Assertions:

- A JSON report is created at the specified path.
- The report exists in the specified location.
- The schema version of the report is correct (1.0).
- The summary statistics are accurate: total=2, passed=1, 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	58 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319, 321-323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	191 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151,

	153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-364, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
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` 62ms 13

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report for a provider enabled.

Why Needed: Prevent regressions by ensuring LLM annotations are present in the report.

Key Assertions:

- The scenario 'Checks the happy path' is present in the report.
- The reason 'Prevents regressions' is true.
- The key assertions 'asserts True' are present in the LLM annotation.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	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	59 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-287, 289, 291-293, 298, 305-308, 310-313, 319, 321-323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	211 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236-238, 240-241, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-366, 369, 371, 374-378, 381, 383-388, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
src/pytest_llm_report/prompts.py	29 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-109, 111-112, 116)
src/pytest_llm_report/report_writer.py	105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-296, 298-299, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-

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

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_llm_error_is_reported` 88ms 13

AI ASSESSMENT

Scenario: Verify that LLM errors are surfaced in HTML output.

Why Needed: Prevents regression where LLM errors are not reported correctly.

Key Assertions:

- Asserts the presence of 'LLM error' and 'boom' in the report content.
- Verifies that the error message contains the word 'boom'
- Checks for the correct file path to the report output.
- Ensures that the LLM error is reported correctly even with a custom model.
- Verifies that the error message includes the expected error type and message.
- Checks if the content of the report matches the expected format.
- Ensures that the test can reproduce the issue even when running multiple tests in parallel.

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	35 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, 205-206, 208)

src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	59 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-287, 289, 291-293, 298, 305-308, 310-313, 319-321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	211 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236-238, 240-241, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-366, 369, 371, 374-379, 383-388, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
src/pytest_llm_report/prompts.py	29 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-109, 111-112, 116)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	101 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-296, 298-299, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

PASSED tests/test_smoke_pytester.py::TestMarkers::test_llm_opt_out_marker 60ms 7

AI ASSESSMENT

Scenario: Test the LLM opt-out marker functionality.

Why Needed: Prevents regression in LLM opt-out marker recording.

Key Assertions:

- The test verifies that the LLM opt-out marker is recorded correctly.
- The test ensures that the LLM opt-out marker is set to False for all tests.
- The test checks if the LLM opt-out marker is properly recorded in the report.json file.
- The test asserts that there is only one test with the LLM opt-out marker in the report.json file.
- The test verifies that the LLM opt-out marker is correctly set to False for all tests.
- The test checks if the LLM opt-out marker is not set to True for any test.
- The test ensures that the LLM opt-out marker does not interfere with other markers in the report.json file.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181-182, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	76 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186-188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	58 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319, 321-323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	191 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71,

	73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-364, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
src/pytest_llm_report/report_writer.py	105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestMarkers::test_requirement_marker` 58ms 7

AI ASSESSMENT

Scenario: Verify that the requirement marker is recorded and correctly identified.

Why Needed: This test prevents a potential regression where the requirement marker might not be properly recorded or identified, potentially leading to incorrect reporting of tests.

Key Assertions:

- The `@pytest.mark.requirement` annotation should record the specified requirements.
- The `requirement` attribute of the test function should contain the required values.
- The `REQ-001` and `REQ-002` strings should be present in the `requirements` list of the test function.
- The `REQ-001` string should be found in the `requirements` list of the first test function.
- The `REQ-002` string should be found in the `requirements` list of the first test function.
- The `pytester.makepyfile()` function should create a file with the specified requirements.
- The `pytester.runpytest()` function should run the tests and generate a report.
- The generated report should contain the correct number of tests (1 in this case).
- The JSON file containing the test report should have the expected structure.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 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	58 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319, 321-323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)

src/pytest_llm_report/plugin.py	191 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-364, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
src/pytest_llm_report/report_writer.py	105 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_smoke_pytester.py::TestOutcomes::test_multiple_xfail_outcomes` 66ms 7

AI ASSESSMENT

Scenario: The test verifies that multiple xfailed tests are recorded in the report.

Why Needed: This test prevents regression by ensuring that all xfailed tests are properly reported and counted.

Key Assertions:

- 2
- ['xfailed', 'xfailed']

COVERAGE

src/pytest_llm_report/collector.py	47 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-116, 119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212, 216, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	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	58 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319, 321-323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	191 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208,

	210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-364, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
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 59ms 7

AI ASSESSMENT

Scenario: Test that skipping tests prevents the 'skip' marker from appearing in the report.

Why Needed: This test ensures that skipping tests is properly recorded and reported, preventing false positives.

Key Assertions:

- The 'summary' key in the report JSON should contain a count of skipped tests.
- The value of the 'skipped' key should be equal to 1.
- The 'skip' marker should not appear in the report.

COVERAGE

src/pytest_llm_report/collector.py	43 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 106-107, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	76 lines (ranges: 162, 166-171, 173-175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	58 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319, 321-323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	191 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184,

	186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-364, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
src/pytest_llm_report/report_writer.py	107 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317, 319, 321-322, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED tests/test_smoke_pytester.py::TestOutcomes::test_xfail_outcome 62ms 7

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	47 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-116, 119, 121-122, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 212, 216, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	76 lines (ranges: 162, 166-171, 173-175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	58 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319, 321-323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	191 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-364, 391-401, 413-414, 417, 421-423,

	434, 438, 457, 461-463, 474, 478, 481, 483-484)
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` 63ms 7

AI ASSESSMENT

Scenario: Test the parameterized tests feature.

Why Needed: This test prevents a regression that could occur when using the --llm-report-json flag with pytester.

Key Assertions:

- The total number of successful parametrized tests is 3.
- All parametrized tests passed successfully.
- Each parametrized test was run only once.
- No duplicate test runs were performed.
- No test failures occurred during the run.
- No unexpected errors occurred during the run.
- The report generated by pytester contains accurate information about the tests.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163-164, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	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	58 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319, 321-323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	191 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99,

	101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 271, 275, 279, 282, 301-302, 309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-364, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
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 72ms 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	57 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319, 321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	143 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 413-414, 417, 421-423)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_markers_registered` 46ms 3

AI ASSESSMENT

Scenario: Test that LLM markers are registered and correctly displayed in the pytest output.

Why Needed: This test prevents a bug where LLM markers are not correctly registered or displayed in the pytest output, potentially leading to false positives or incorrect marker usage.

Key Assertions:

- The 'llm_opt_out' marker should be found in the stdout of the pytest run.
- The 'llm_context' marker should be found in the stdout of the pytest run.
- The 'requirement' marker should be found in the stdout of the pytest run.
- The 'llm_opt_out' marker should match the expected output line.
- The 'llm_context' marker should match the expected output line.
- The 'requirement' marker should match the expected output line.
- All three markers should have different line numbers in their stdout output.
- The 'llm_opt_out' marker should not be present in the stdout of a test that does not use this marker.
- The 'llm_context' marker should not be present in the stdout of a test that does not use this marker.
- The 'requirement' marker should not be present in the stdout of a test that does not use this 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/options.py`	`57 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319, 321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)`

src/pytest_llm_report/plugin.py	143 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 413-414, 417, 421-423)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_plugin_registered` 53ms 3

AI ASSESSMENT

Scenario: Verify that the plugin is registered correctly by running pytest with --help flag.

Why Needed: This test prevents a potential issue where the plugin is not registered due to an incorrect or missing configuration.

Key Assertions:

- The `--llm-report` flag should be present in the output of `pytester.runpytest('--help')`.
- The `--llm-report` flag should match the expected output for a successful plugin registration.
- The test should fail if the `--llm-report` flag is not present or does not match the expected output.
- The test should pass if the `--llm-report` flag is correctly configured and 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/options.py	57 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319, 321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	143 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 413-414, 417, 421-423)

PASSED `tests/test_smoke_pytester.py::TestSpecialCharacters::test_special_chars_in_nodeid` 88ms 8

AI ASSESSMENT

Scenario: Verify that special characters in nodeid are handled correctly by pytester.

Why Needed: This test prevents a potential crash and ensures the HTML generated is valid.

Key Assertions:

- The string "hello" should be parsed as a single string.
- The string "foo&bar" should be parsed as a single string.
- The HTML file should contain the '' tag.
- The report path should exist and contain the valid HTML content.
- The report path should not crash when run with pytester.
- The generated HTML should have a valid structure.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163-164, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	58 lines (ranges: 123, 163, 191, 194-195, 201-202, 209-210, 217-218, 225-226, 233, 237, 239, 241, 243, 245, 248, 252, 276, 279-281, 283-287, 289, 291-293, 298, 305-308, 310-313, 319-321, 323, 325, 327, 329, 331, 335, 337, 339, 341, 343, 347, 349)
src/pytest_llm_report/plugin.py	191 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 160-163, 165-168, 170-173, 175-179, 181-184, 186-189, 202-204, 206-208, 210-212, 216, 220-221, 223, 225, 228-229, 236, 245-246, 271, 275, 279, 282, 301-302,

	309-310, 313-314, 316-317, 320-324, 326, 329-330, 332, 335-336, 357, 363-364, 391-401, 413-414, 417, 421-423, 434, 438, 457, 461-463, 474, 478, 481, 483-484)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	101 lines (ranges: 55, 67-73, 85-86, 98-100, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

📄 tests/test_time.py

15 tests

PASSED tests/test_time.py::TestFormatDuration::test_boundary_one_minute 1ms 3

AI ASSESSMENT

Scenario: Tests the 'format_duration' function with a boundary of exactly one minute.

Why Needed: This test prevents regression in case the input duration is greater than or equal to one minute.

Key Assertions:

- The output should be '1m 0.0s'.
- The number of decimal places should be zero.
- The format string should include a unit ('m') and an optional sign ('+' or '-').
- The duration value should not exceed one minute (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: 413-414, 417, 421-423)
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: Test formats sub-millisecond durations as microseconds.

Why Needed: Prevents a potential bug where the test fails due to incorrect formatting of microsecond durations.

Key Assertions:

- The function `format_duration(0.0005)` should return '500µs' when given an input of 0.0005 seconds.
- The string representation of the output should contain exactly '500µs'.
- The unit 'µs' should be present in the output string.
- The function `format_duration()` is correctly called with a non-zero argument (0.0005).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/time.py	2 lines (ranges: 39-40)

PASSED tests/test_time.py::TestFormatDuration::test_milliseconds_format 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function to ensure it correctly formats sub-second durations as milliseconds.

Why Needed: This test prevents a potential bug where the function does not format the duration correctly for millisecond precision.

Key Assertions:

- The result of `format_duration(0.5)` is expected to contain 'ms' in its string representation.
- The formatted string should be equal to '500.0ms'.
- The function should handle sub-second durations (e.g., 0.25, 0.75) correctly and format them as milliseconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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: Tests the 'minutes_format' function to ensure it correctly formats durations over a minute.

Why Needed: This test prevents a potential regression where the duration is misinterpreted as being in seconds instead of minutes.

Key Assertions:

- The result contains the string 'm' which indicates that the duration is in minutes.
- The result equals the expected string '1m 30.5s' which represents the correct format for a duration over a minute.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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: Test formats multiple minutes to include both minutes and seconds.

Why Needed: Prevents regression in formatting of duration with multiple minutes.

Key Assertions:

- The function `format_duration(185.0)` correctly formats the input as '3m 5.0s'.
- The result does not exceed the maximum allowed length for a duration string (100 characters).
- The function handles cases where the input is less than or equal to 1 minute.
- The function handles cases where the input is greater than or equal to 60 minutes.
- The function correctly handles decimal values in minutes.
- The function preserves the original order of digits for both minutes and seconds.
- The function does not introduce any new formatting rules that would break existing tests.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

PASSED tests/test_time.py::TestFormatDuration::test_one_second 1ms 3

AI ASSESSMENT

Scenario: Verifies the correct formatting of a duration equal to one second.

Why Needed: To ensure that the `format_duration` function returns the expected string representation for durations exactly equal to one second.

Key Assertions:

- The result should be '1.00s' when the input is 1.0.
- The result should not include any trailing zeros.
- The result should have a decimal point immediately after the number.
- The result should only contain two digits after the decimal point.
- No leading zeros are allowed in the result.
- No negative numbers are allowed as inputs.
- The function should handle cases where the input is 0.0 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: 413-414, 417, 421-423)
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: Verify that the function correctly formats seconds under a minute.

Why Needed: This test prevents a potential bug where the function does not format seconds correctly when they are less than one minute.

Key Assertions:

- The result should contain the string 's' to indicate seconds.
- The result should be equal to '5.50s' to match the expected output.
- The function should handle cases where the input is a fraction of a second correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED tests/test_time.py::TestFormatDuration::test_small_milliseconds 1ms 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 1 millisecond.

Why Needed: This test prevents a potential bug where the function incorrectly returns '1.0ms' for durations less than 1 millisecond.

Key Assertions:

- The function should return the correct value for a duration of 1 millisecond (1.0ms).
- The function should handle cases where the input is negative or zero correctly.
- The function should not return 'nan' (Not a Number) when the input is infinity.
- The function should preserve the original precision of the input value.
- The function should raise an error for invalid input types (e.g., non-numeric values).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/time.py	3 lines (ranges: 39, 41-42)

PASSED tests/test_time.py::TestFormatDuration::test_very_small_microseconds 1ms 3

AI ASSESSMENT

Scenario: Verifies that the function correctly formats very small durations as microseconds.

Why Needed: This test prevents a potential bug where the function incorrectly formats very small durations as nanoseconds or milliseconds.

Key Assertions:

- The result of `format_duration(0.000001)` is equal to '1µs'.
- The duration is formatted correctly as microseconds.
- The duration is not formatted as nanoseconds or milliseconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/time.py	2 lines (ranges: 39-40)

PASSED tests/test_time.py::TestIsoFormat::test_formats_datetime_with_utc 1ms 3

AI ASSESSMENT

Scenario: Tests the `iso_format` function with a datetime object representing UTC time.

Why Needed: Prevents regression in handling datetime objects with UTC timezone.

Key Assertions:

- The output of `iso_format(dt)` is '2024-01-15T10:30:45+00:00'.
- The input `dt` has a valid UTC timezone.
- The function correctly formats the datetime object as per ISO 8601 standard.
- The formatted string does not contain any time zone offset information.
- No exceptions are raised when passing an invalid datetime object with UTC timezone.
- The function handles daylight saving time (DST) correctly by preserving it in the output.
- The input `dt` is a valid datetime object that can be used to create an ISO formatted 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: 413-414, 417, 421-423)
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: Verify that naive datetime is formatted correctly without timezone.

Why Needed: Prevents a potential bug where the naive datetime format is incorrect due to missing timezone information.

Key Assertions:

- The output of `iso_format(dt)` should be '2024-06-20T14:00:00'.
- The output does not include any timezone information.
- The output is in the correct ISO 8601 format.
- The output does not contain any invalid characters or formatting errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

PASSED `tests/test_time.py::TestIsoFormat::test_formats_with_microseconds` 1ms 3

AI ASSESSMENT

Scenario: Tests the `iso_format` function with a datetime object containing microseconds.

Why Needed: Prevents a bug where microseconds are not included in the formatted ISO string.

Key Assertions:

- The 'result' variable should contain the string '123456'.
- The 'result' variable should contain the substring '123456' (case-insensitive).
- The 'result' variable should be a string containing only digits.
- The 'result' variable should not be empty.
- The 'result' variable should not contain any non-digit characters.
- The 'result' variable should be a valid ISO 8601 format string.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

PASSED `tests/test_time.py::TestUtcNow::test_has_utc_timezone` 1ms 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 the function returns a current time within a specified tolerance.

Why Needed: This test prevents a potential issue where the function does not return a current time that is within the expected range for UTC.

Key Assertions:

- The function `utc_now()` returns a datetime object representing the current time in UTC.
- The difference between `before` and `result` should be less than or equal to `after - result` (within a tolerance of 1 second).
- The difference between `after` and `result` should be greater than `before - result` (less than or equal to 1 second).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)

PASSED tests/test_time.py::TestUtcNow::test_returns_datetime 1ms 3

AI ASSESSMENT

Scenario: The function `utc_now()` returns a datetime object.

Why Needed: This test prevents a potential issue where the function might return an incorrect datetime object if the system clock is not properly synchronized.

Key Assertions:

- The result of `utc_now()` should be a datetime object.
- The result of `utc_now()` should have a valid timezone.
- The result of `utc_now()` should have a valid date and time.
- The result of `utc_now()` should not raise any exceptions.
- The result of `utc_now()` should be a valid datetime object with the correct timezone.
- The result of `utc_now()` should have a timezone that is compatible with the system's timezone.
- The result of `utc_now()` should not return an empty datetime object.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)
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 is executed with an invalid refresh interval.

Why Needed: This test prevents a potential bug where the TokenRefresher class does not handle cases where the provided refresh interval exceeds the maximum allowed value, leading to unexpected behavior or errors.

Key Assertions:

- The 'refresh_interval' attribute of the TokenRefresher instance is set to 3600 seconds.
- The 'output_format' attribute of the TokenRefresher instance is set to 'text'.
- When the 'get-token' method is called with an invalid refresh interval (e.g., less than 1 second), it raises a TokenRefreshError exception with an error message indicating authentication failed.
- The error message includes the string 'exit 1', which indicates that the command failed and returned a non-zero exit code.
- The error message includes the string 'Authentication failed', which is the expected error message for this 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/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: 413-414, 417, 421-423)`

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_empty_output 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, 107-109, 113, 115)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_force_refresh 1ms 3

AI ASSESSMENT

Scenario: Test the 'force_refresh' feature of TokenRefresher.

Why Needed: This test prevents a potential bug where the cache is not updated when the refresh interval is set to a value that does not require a new token.

Key Assertions:

- Verify that the correct token is returned after calling `get_token()` with `force=True`.
- Verify that the second call to `get_token()` returns a different token than the first one.
- Verify that the number of calls to `get_token()` increases by 1 when `force=True` is used.
- Verify that the output format is set to 'text' as expected.
- Verify that the error message is empty as expected.
- Verify that the return code is 0 as expected.
- Verify that the stdout contains a token with the call count.
- Verify that the stderr is empty 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/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: 413-414, 417, 421-423)

PASSED `tests/test_token_refresh.py::TestTokenRefresher::test_get_token_json_custom_key` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the TokenRefresher function correctly retrieves a custom JSON key for accessing an access token.

Why Needed: This test prevents a bug where the custom JSON key is not properly retrieved from the subprocess call to get-token.

Key Assertions:

- - The output of the subprocess call contains the expected custom JSON key.
- - The 'access_token' field in the response matches the custom JSON key.
- - The error message is empty, indicating that no exception was raised during the subprocess call.
- - The 'stdout' attribute of the subprocess result object contains the custom JSON key as a string.
- - The 'json_key' parameter passed to the TokenRefresher constructor matches the custom JSON key.
- - The 'output_format' parameter is set to 'json', which allows for the correct parsing of the custom JSON key.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/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: 413-414, 417, 421-423)`

PASSED `tests/test_token_refresh.py::TestTokenRefresher::test_get_token_json_format` 1ms 3

AI ASSESSMENT

Scenario: Verify the `TokenRefresher` extracts a JSON token from the expected output.

Why Needed: This test prevents a potential bug where the `get-token` command returns an incorrect or incomplete JSON response, potentially causing the `TokenRefresher` to fail or produce unexpected results.

Key Assertions:

- The `token` key in the output should contain the expected value 'json-token-value'.
- The `expires_in` value should be set to 3600 (1 hour) as specified by the `refresh_interval` parameter.
- The JSON response should not include any other keys or values than the required ones.
- The `stdout` output of the `fake_run` function should contain a JSON string with the expected token and expiration time.
- The `stderr` output of the `fake_run` function should be empty.
- The `json.dumps` function should return a valid JSON string containing the specified keys and values.
- The `returncode` of the `fake_run` function should be 0, indicating successful execution.
- The `output_format` parameter should be set to 'json' 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/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: 413-414, 417, 421-423)

PASSED `tests/test_token_refresh.py::TestTokenRefresher::test_get_token_text_format` 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the `TokenRefresher` class extracts the correct text format from the output of the `get-token` command.

Why Needed: This test prevents a bug where the extracted token is not in the expected text format, potentially leading to incorrect usage or downstream errors.

Key Assertions:

- The output of the `get-token` command contains the string "my-secret-token".
- The output does not contain any non-text characters (e.g., newline, tab).
- The extracted token is in lowercase.
- The extracted token has a length greater than 50 characters.
- The extracted token starts with 'my-'.
- The extracted token contains only alphanumeric characters and underscores.
- The extracted token does not contain any whitespace characters (spaces, tabs, etc.).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: 413-414, 417, 421-423)

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_invalid_json 1ms 3

AI ASSESSMENT

Scenario: Test that TokenRefresher raises an error on invalid JSON input.

Why Needed: This test prevents a potential bug where the TokenRefresher class does not handle invalid JSON inputs correctly.

Key Assertions:

- The `get_token` method of `TokenRefresher` should raise a `TokenRefreshError` with a message indicating that the input is invalid JSON.
- The error message should include the word 'json' to ensure it's a valid JSON string.
- The test should fail when an invalid JSON string is passed as input, and the error message should be clear about this.

COVERAGE

src/pytest_llm_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: 413-414, 417, 421-423)

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_invalidate 1ms 3

AI ASSESSMENT

Scenario: Test that the `invalidate` method of `TokenRefresher` clears its cache and updates the token count correctly.

Why Needed: This test prevents a potential issue where the `invalidate` method does not clear the cache, leading to stale tokens being returned.

Key Assertions:

- The function `invalidate()` of `TokenRefresher` should set the `call_count` variable to 2 after calling it with the provided arguments.
- The function `get_token()` of `TokenRefresher` should return a token that is different from the one obtained before calling `invalidate()`.
- The function `invalidate()` of `TokenRefresher` should clear its cache by setting `call_count` to 0 after calling it with the provided arguments.
- The function `get_token()` of `TokenRefresher` should return a token that is different from the one obtained before calling `invalidate()` and has a different value than the previous token.
- The function `invalidate()` of `TokenRefresher` should not have any side effects other than updating the cache and token count.
- The function `get_token()` of `TokenRefresher` should return a token that is different from the one obtained before calling `invalidate()` and has a different value than the previous token.
- The function `invalidate()` of `TokenRefresher` should not modify any external state other than updating the cache and token count.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: 413-414, 417, 421-423)

PASSED `tests/test_token_refresh.py::TestTokenRefresher::test_missing_json_key` 3ms 3

AI ASSESSMENT

Scenario: Test that TokenRefresher raises an error when the JSON key is missing.

Why Needed: To prevent a potential bug where the TokenRefresher class does not raise an error when the required JSON key is missing from the token response.

Key Assertions:

- The 'token' key should be present in the output of the get_token method.
- The 'not found' message should be present in the output of the get_token method.
- The 'token' key should not be present in the error message.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	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: 413-414, 417, 421-423)

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 all threads get the same token.

Why Needed: This test prevents a potential bug where multiple threads accessing the TokenRefresher instance simultaneously could result in inconsistent or incorrect token acquisition.

Key Assertions:

- The function `get_token()` should acquire the lock before returning the token.
- All threads should get the same token (first one to acquire lock).
- The length of the set of results should be equal to 1.
- The first element in the list of results should be 'token-1'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	28 lines (ranges: 59-60, 63-66, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132, 153-154, 156)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_timeout_handling 1ms 3

AI ASSESSMENT

Scenario: The test verifies that the TokenRefresher handles command timeouts correctly by raising a TokenRefreshError when the 'get-token' command takes longer than 30 seconds to complete.

Why Needed: This test prevents a potential bug where the TokenRefresher does not raise an error when it encounters a timeout, potentially causing unexpected behavior or data loss.

Key Assertions:

- The function will raise a `TokenRefreshError` with the message 'timed out' when the command takes longer than 30 seconds to complete.
- The function will set `exc_info.value` to an instance of `TokenRefreshError` with the message 'timed out'.
- The function will assert that the error message contains the substring 'timed out'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	16 lines (ranges: 59-60, 63, 69, 83, 85-86, 90, 93-98, 113-114)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)

PASSED tests/test_token_refresh.py::TestTokenRefresher::test_token_caching 1ms 3

AI ASSESSMENT

Scenario: Test Token Caching: Verify that the TokenRefresher caches tokens and doesn't call command again.

Why Needed: This test prevents a potential bug where the TokenRefresher calls the command multiple times due to caching, leading to unnecessary computations.

Key Assertions:

- The function `get_token()` of the `TokenRefresher` class returns the same token for both `token1` and `token2`.
- The value of `call_count` is equal to 1 after calling `get_token()` twice.
- Both `token1` and `token2` have the same value 'token-1'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/token_refresh.py	28 lines (ranges: 59-60, 63-66, 69-72, 83, 85-86, 90, 93-98, 101, 107-108, 111, 132, 153-154, 156)
src/pytest_llm_report/plugin.py	6 lines (ranges: 413-414, 417, 421-423)