Test Report

Run ID: 21100712177-py3.12 • Generated: 2026-01-17 20:55:33 • Duration: 16.56s
Plugin: v0.1.0 (b7a157f6cb9189cc50a17c846484c8454deeac61) [dirty]
Repo: v0.1.1 (c02209002d41f9d884005e252e8a20735f87b081)
LLM: ollama / llama3.2:1b (minimal context, 350 annotated, 36 errors)

90.52%
Total Coverage

387
TOTAL TESTS

387
PASSED

0
FAILED

0
SKIPPED

0
XFAILED

0
XPASSED

0
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	116	5	111	95.69%	13, 15-19, 21, 35, 38, 44, 46, 52-53, 55-57, 59, 61-64, 69, 73-74, 77-80, 84, 87-89, 93, 103, 109-111, 113-117, 119-120, 125, 127-128, 130-131, 134-135, 141-144, 146, 148, 162, 164, 168, 170, 172, 182, 184-188, 190-191, 194, 196, 205, 217, 219-233, 235, 237, 245-246, 248-249, 251, 253-255, 259, 262-263, 265-266, 269, 271-272, 274, 276-277, 281	66, 90-91, 192, 203
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,	141, 239

					271, 277, 279, 285	
src/pytest_llm_report/coverage_map.py	135	10	125	92.59%	13, 15-17, 19-22, 30, 38, 44-45, 47, 58-60, 64, 72-73, 83, 86, 88-90, 92, 94-96, 98, 101-104, 106-108, 114, 116, 118, 121-122, 127, 131-135, 137-140, 144-146, 148, 150, 152-153, 156, 160-162, 165, 167-168, 173, 176, 178-184, 187-189, 191, 196, 199-200, 202, 204, 216-217, 220, 224-225, 228-234, 236, 239, 241, 243-244, 246-248, 250, 252-254, 259-260, 263-264, 271, 273, 276-279, 281-283, 285, 299-300, 302, 308	62, 123, 125, 128, 157, 221, 249, 251, 257, 274
src/pytest_llm_report/errors.py	35	0	35	100.0%	8-9, 12, 25-28, 31-36, 39-42, 45-46, 49-51, 54-55, 64-66, 68, 70, 74-76, 80, 129, 139	-
src/pytest_llm_report/llm/__init__.py	3	0	3	100.0%	4-5, 7	-
src/pytest_llm_report/llm/annotator.py	110	0	110	100.0%	4, 6-10, 12-15, 21-22, 25-28, 31, 45-46, 48-50, 54, 56-57, 59, 61-62, 64, 66-68, 71-72, 74-82, 87, 97-98, 100, 102, 104-105, 115, 127, 129-132, 137-139, 142, 165-168, 170-171, 176, 178, 180-183, 185-190, 192-193, 198-201, 203, 206, 229-232,	-

					234, 236-237, 239-240, 245-246, 248-253, 255-256, 261-264, 266	
src/pytest_llm_report/llm/base.py	78	0	78	100.0%	13, 15-18, 26, 40, 46, 52-53, 55, 72, 75-76, 78, 80, 101, 107-108, 110-111, 122, 128, 130, 136, 138, 147, 149, 165, 167-173, 175, 177, 186-187, 190-192, 194-195, 198-200, 203-208, 212, 214, 220-221, 224-225, 228-230, 233, 245, 247, 249-250, 252-253, 255, 257-258, 260, 262-263, 265, 267	-
src/pytest_llm_report/llm/gemini.py	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,	89, 104, 106, 115-117, 199, 228-229, 233, 239-241, 248, 251-254, 256, 262, 373, 447, 450

					395, 397-398, 402-408, 411, 414-416, 418-420, 422-427, 433-434, 436-440, 443-446, 448-449, 451-453	
src/pytest_llm_report/llm/litellm_provider.py	34	6	28	82.35%	7, 9, 11-12, 18, 21, 37-38, 44, 46, 49, 51-52, 54-56, 66-67, 69, 71, 76, 78, 80, 82, 93, 99-100, 102	72, 83-84, 86-87, 89
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/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, 281, 283, 287-	178, 189, 191, 193, 466, 519, 521, 523, 525, 527

				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	117	45	72	61.54%	106, 146, 175, 178-180, 185-187, 193-195, 201-203, 209-218, 220, 224, 233, 248, 251-267, 270-283, 286-295, 298, 300	13-15, 21-22, 90-94, 97-99, 102-105, 122-123, 126-132, 135-137, 140-142, 145, 156-160, 163-164, 167, 169, 222, 227, 236
src/pytest_llm_report/plugin.py	151	24	127	84.11%	40, 43, 49, 55, 61, 67, 73, 80, 89, 95, 101, 107, 113, 121, 126, 131, 136, 142, 147, 153, 169, 173, 177, 183-184, 187-188, 190, 192, 195-197, 203-204, 212-213, 238-239, 242-243, 246, 249-250, 252-253, 256-257, 259, 261-265, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-305, 307, 309-312, 315-316, 324-325, 330-333, 336, 338, 341-346, 348, 350, 358-359, 380-381, 384-385, 388-390, 401-402, 405, 408-409, 412-414, 424-425, 428-430, 441-442, 445, 448, 450-451	13, 15-17, 19-20, 22, 28-31, 34, 160, 216, 320-321, 326-327, 372-373, 393, 417, 433-434

src/pytest_llm_report/prompts.py	75	5	70	93.33%	13, 15-17, 24, 27, 33, 35, 49, 52, 55, 58-61, 63, 65, 67, 78-79, 82-84, 86-87, 92, 94-95, 98-101, 103-112, 116, 118, 132-133, 135-138, 140-141, 144-145, 148, 151-152, 154-156, 158-159, 163, 165, 180, 182, 191-194	80, 114, 142, 146, 149
src/pytest_llm_report/render.py	50	0	50	100.0%	13, 15-16, 18, 24, 30-31, 34, 40, 42, 50-51, 53, 56, 65-67, 70, 79, 87, 90, 99, 101-102, 107, 110, 121-124, 126-129, 131-134, 141-143, 145, 158-163, 177, 196	-
src/pytest_llm_report/report_writer.py	167	10	157	94.01%	13, 15-25, 27-29, 46, 55, 58, 67-68, 76, 83-84, 89, 98-100, 102, 105-108, 110, 116, 127-128, 130, 142, 150, 156-158, 160, 186-189, 192, 197-199, 202-203, 211, 222-223, 226-227, 230-231, 233, 235, 254, 256-259, 262-264, 266, 268, 303, 312, 314-315, 317-328, 330, 332, 340, 343-345, 348-349, 352-354, 357, 360, 368, 376, 378-379, 382, 385, 388, 391, 399, 401-402, 408, 410, 412, 414-423, 434-435, 437-439, 447-448, 453, 455, 458, 461-462, 464, 470-474, 480-481, 488, 495, 497,	113, 135-137, 424-425, 432, 449-451

					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 reports to ensure both tests are included in the aggregated report.

Why Needed: This test prevents regression where only one or no test cases are included in the aggregated report.

Key Assertions:

- The aggregated report should include both retained test case from each individual report.
- The number of tests in the aggregated report should be equal to the total number of tests in both reports.
- Each retained test case should be present in the aggregated report.
- If a test is not included in the aggregated report, it should not have any associated reports.
- If a test has no associated reports, its duration and phase should be empty strings.
- The aggregated report should contain all nodes with their respective outcomes.
- Each node's outcome should match the corresponding outcome of the retained tests.
- The aggregated report should contain the total number of tests as specified in the test case.

COVERAGE

`src/pytest_llm_report/aggregation.py`	`69 lines (ranges: 52, 55-56, 59, 61-63, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 130-131, 134, 141, 146, 148-153, 155, 157-159, 170, 217, 219-223, 235, 245, 248-249, 251, 253, 276-279, 281)`
`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the `aggregate` function throws an error if the specified directory does not exist.

Key Assertions:

- The `aggregator.aggregate()` method is called with no arguments.
- No exception is raised.
- The `pathlib.Path.exists()` mock returns False.
- The `pathlib.Path.exists()` mock is called only once.
- The `aggregate` function does not throw an error when the directory exists.
- The `aggregate` function throws a `FileNotFoundError` when the directory exists.
- The `aggregate` function does not raise any exception even if the directory does not exist.

COVERAGE

src/pytest_llm_report/aggregation.py	7 lines (ranges: 52, 55-57, 109-111)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the aggregator picks the latest policy for aggregate aggregation

Why Needed: This test prevents regression in case of simultaneous runs with different times.

Key Assertions:

- The aggregated report from `report2` is considered as the latest.
- The outcome of `test_case` is marked as 'failed' in `report1` and passed in `report2`.
- The run meta contains both aggregated results for `run1` and `run2` with 2 runs each.
- The summary indicates that there was only one test passed (from `report2`).
- Only the outcome of `test_case` is marked as 'failed' in `report1`.

COVERAGE

src/pytest_llm_report/aggregation.py	77 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 130-131, 134, 141, 146, 148-153, 155, 157-159, 170, 182, 184-188, 190-191, 194, 217, 219-223, 235, 245, 248-249, 251, 253, 276-279, 281)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 regression in case of missing aggregation directory configuration.

Key Assertions:

- The aggregate function should return None for the given mock_config.
- The aggregate method should be called with an empty string as its argument.
- No exception should be raised when calling aggregate() on a non-existent directory.
- The test should fail if agg.aggregate() is called with a valid configuration.
- The test should pass if agg.aggregate() is called with an invalid configuration (e.g., None or empty string)
- The test should verify that the aggregate method does not raise any exceptions when called on a non-existent directory.
- The test should verify that the aggregate method returns None for a valid aggregation function.

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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the `aggregate` method returns `None` when no reports exist.

Why Needed: This test prevents a potential bug where an empty report directory leads to incorrect aggregation results.

Key Assertions:

- aggregator.aggregate() should return None for an empty report directory.
- aggregator.aggregate() should not raise any exceptions for an empty report directory.
- the `aggregate` method should be able to handle the case when no reports exist.
- an empty report directory should not prevent aggregation from working correctly.

COVERAGE

`src/pytest_llm_report/aggregation.py`	9 lines (ranges: 52, 55-57, 109-110, 113-114, 170)
`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: Prevents regression in core functionality due to incorrect serialization of LLM annotations.

Key Assertions:

- Coverage was properly deserialized with correct file paths and line counts.
- LLM annotation was properly deserialized with correct scenario, why needed, and key assertions.
- Serialized report matches original report after fix.

COVERAGE

src/pytest_llm_report/aggregation.py	81 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 130-131, 134-137, 141-144, 146, 148-153, 155, 157-159, 170, 182, 184-188, 194, 217, 219-223, 235, 245, 248-249, 251, 253, 276-279, 281)
src/pytest_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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify source coverage summary deserialization.

Why Needed: Prevents a potential bug where the source coverage is not properly deserialized, potentially leading to incorrect analysis or reporting.

Key Assertions:

- The 'source_coverage' key in the report should be an array of SourceCoverageEntry objects.
- Each SourceCoverageEntry object should have the following properties: 'file_path', 'statements', 'missed', 'covered', 'coverage_percent', 'covered_ranges', and 'missed_ranges'.
- The 'coverage_percent' property should be a number between 0 and 100.
- The 'covered_ranges' property should contain strings representing ranges of covered code (e.g. '1-5, 7-11').
- The 'missed_ranges' property should contain strings representing ranges of missed code (e.g. '6, 12').
- Each SourceCoverageEntry object should have a 'file_path' attribute that matches the file path in the report.
- The aggregated result should not be None and should contain exactly one SourceCoverageEntry object.

COVERAGE

src/pytest_llm_report/aggregation.py	66 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119, 125, 127-128, 148-155, 157-159, 170, 182, 184-186, 194, 217, 219-220, 235, 245, 248-249, 251, 253, 276-279, 281)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- Verify that `_load_coverage_from_source()` returns `None` when `llm_coverage_source` is `None`.
- Verify that `_load_coverage_from_source()` raises a `UserWarning` when `llm_coverage_source` is `/nonexistent/coverage`.
- Verify that `_load_coverage_from_source()` successfully loads coverage data from the configured source file (`.coverage`).

COVERAGE

src/pytest_llm_report/aggregation.py	19 lines (ranges: 245-246, 248-249, 251, 253-257, 259, 262-263, 265-266, 269, 271-272, 274)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: test_recalculate_summary verifies that the aggregator recalculates the summary correctly when given a list of test results and a mock latest summary.

Why Needed: This test prevents regression where the aggregator does not recalculate the summary correctly when given incomplete or outdated information.

Key Assertions:

- The total count of tests passed should be equal to the provided number of tests.
- The number of failed tests should be equal to the provided number of failed tests.
- The number of skipped tests should be equal to the provided number of skipped tests.
- The number of xfailed tests should be equal to the provided number of xfailed tests.
- The number of xpassed tests should be equal to the provided number of xpassed tests.
- The error count should be equal to the provided number of errors.
- The coverage percentage should remain preserved after recalculating the summary.

COVERAGE

`src/pytest_llm_report/aggregation.py`	17 lines (ranges: 217, 219-233, 235)
`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test ensures that the `aggregate` function behaves correctly when it encounters an invalid JSON file.

Key Assertions:

- The `aggregate` function skips the 'invalid.json' report.
- Only valid reports are counted in the aggregate result.
- Missing fields in the 'invalid.json' report do not affect the aggregation count.

COVERAGE

src/pytest_llm_report/aggregation.py	71 lines (ranges: 52, 55-56, 59, 64, 69, 73-74, 77-80, 84, 87-89, 93-100, 109-110, 113-117, 119-120, 125, 127-128, 148-153, 155, 157-159, 162, 164-166, 168, 170, 182, 184-186, 194, 217, 219-220, 235, 245, 248-249, 251, 253, 276-279, 281)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

📄 tests/test_aggregation_maximal.py

1 tests

PASSED `tests/test_aggregation_maximal.py::TestAggregationMaximal::test_recalculate_summary_coverage` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the aggregator recalculates the summary with correct total duration and coverage percentage when given a list of tests and a latest summary.

Why Needed: This test prevents regression where the aggregator fails to calculate the summary correctly after multiple iterations or when the latest summary is not available.

Key Assertions:

- The total duration of the aggregated tests should be equal to the sum of individual test durations.
- The passed count of the aggregated tests should match the number of tests provided in the input list.
- The failed count of the aggregated tests should match the number of tests that failed in the input list.
- The coverage percentage of the aggregated tests should be equal to the coverage percentage of the latest summary.
- The total duration of the aggregated tests should be greater than or equal to the sum of individual test durations.
- The passed count of the aggregated tests should be greater than or equal to the number of tests that failed in the input list.

COVERAGE

`src/pytest_llm_report/aggregation.py`	`10 lines (ranges: 44, 217, 219-225, 235)`
`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

📄 tests/test_annotator.py

7 tests

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

AI ASSESSMENT

Scenario: Testing the caching of tests to prevent skipping

Why Needed: This test prevents a regression where cached tests are skipped due to an incorrect cache implementation.

Key Assertions:

- Mocking `mock_provider` with a valid provider instance is sufficient to skip cached tests.
- Mocking `mock_cache` with a valid cache instance is necessary to prevent skipping cached tests.
- Mocking `mock_assembler` with a valid assembler instance is required for accurate test skipping logic.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-81, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential performance regression where the annotator function may be slowed down by concurrent requests to different providers and caches.

Key Assertions:

- mock_provider.get() should not block for more than 1 second when called concurrently
- mock_cache.get() should return cached results immediately when called concurrently
- annotator function should not take longer than 2 seconds to complete when called concurrently

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	64 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104-112, 129-135, 137, 139, 229-232, 234, 236-237, 239, 245-246, 248-253, 255, 261, 266)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that concurrent annotation handling prevents failures in annotate function

Why Needed: This test verifies that the annotator function is designed to handle concurrent annotations without crashing or producing unexpected results.

Key Assertions:

- mock_provider, mock_cache, and mock_assembler are created with MagicMock instances.
- The annotator function is called concurrently on these mocks.
- The annotator function should not crash or produce an error when handling failures in the mocks.
- The annotator function should be able to handle multiple annotations without issues.
- The annotator function should log any errors that occur during annotation.
- The annotator function should return a success status even if one of the mocks fails.
- The annotator function should not throw an exception when handling failures in the mocks.
- The annotator function should be able to handle multiple annotations concurrently without blocking or waiting for results.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104-112, 129-135, 137-139, 229-232, 234, 236-237, 239, 245-246, 248-253, 255, 261-264, 266)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify progress reporting functionality in the annotator.

Why Needed: This test prevents regression when using progress reporting for large datasets.

Key Assertions:

- The `progress_reporting` method is called with the correct arguments.
- The `progress_reporting` method updates the cache correctly.
- The `progress_reporting` method logs messages as expected.
- The annotator's progress bar updates correctly.
- The cache size is updated correctly after each iteration.
- No exceptions are raised when using progress reporting.
- The progress reporting message is displayed correctly on the console.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verifies the sequential annotation of a test function

Why Needed: Prevents potential regression in sequential annotation testing.

Key Assertions:

- The `test_sequential_annotation` method is called with three mocked dependencies.
- Each mocked dependency is used in the correct order to simulate sequential execution.
- No unexpected side effects are introduced by using mock dependencies in sequence.
- Mocking dependencies in a specific order ensures that tests run sequentially as expected.
- The `test_sequential_annotation` method verifies that the annotator can correctly annotate the test function with sequential dependencies.
- The annotator's behavior is tested to ensure it adheres to the expected sequential annotation pattern.
- The sequence of annotations performed by the annotator is verified and validated for correctness.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The `test_skips_if_disabled` function should be executed without any changes to the test results when the LLM (Language Model Language) is disabled.

Why Needed: This test prevents a regression where the annotator might incorrectly skip tests if the LLM is enabled, leading to incorrect test results or missing test coverage.

Key Assertions:

- The `config` object is created with the provider set to 'none'.
- The `annotate_tests` function is called without any arguments. This means no tests will be annotated.
- No changes are made to the test results when the LLM is disabled.
- The annotator does not skip any tests if it's disabled, ensuring correct test coverage.
- The `test_skips_if_disabled` 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: 107, 147, 224)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- Mocked `provider` instance should be mocked with an error message.
- Captured output from `sys.exit(1)` should contain 'Provider unavailable'.
- The annotator's behavior (skipping tests) should remain unchanged.
- No other test functions should fail due to the provider being unavailable.
- The test function should still return a non-zero exit code.
- The captured output does not indicate an error in the `provider` instance.
- Other test functions should continue running without interruption.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

📄 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 the annotator's behavior when annotating concurrently with progress and errors.

Why Needed: This test prevents regression that may occur when annotating multiple tasks simultaneously, especially if there are errors or timeouts involved.

Key Assertions:

- Verify that the annotator reports both a first error and the correct number of annotated tasks.
- Check that the progress messages include 'Processing X test(s)' for each task.
- Assert that the LLM annotation message includes 'first error'.
- Confirm that there is at least one failure in the annotations.
- Verify that the annotator correctly handles errors and timeouts by reporting them as separate issues.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	28 lines (ranges: 229-232, 234, 236-237, 239-242, 245-246, 248-253, 255-258, 261-264, 266)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: test_annotate_sequential_rate_limit_wait verifies that the annotator waits if the rate limit interval has not elapsed.

Why Needed: This test prevents regression where the annotator does not wait for the rate limit interval to elapse, potentially leading to incorrect annotation results.

Key Assertions:

- assert mock_sleep.called
- assert mock_time.side_effect == [100.0, 100.1, 100.2, 100.3, 100.4]
- assert tasks[0].outcome == 'p' and tasks[0].nodeid == 't1'
- assert tasks[1].outcome == 'p' and tasks[1].nodeid == 't2'
- assert tasks[2].outcome == 'p' and tasks[2].nodeid == 't3'
- assert tasks[3].outcome == 'p' and tasks[3].nodeid == 't4'
- assert tasks[4].outcome == 'p' and tasks[4].nodeid == 't5'
- assert tasks[0].annotation is None
- assert tasks[1].annotation is None
- assert tasks[2].annotation is None

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	23 lines (ranges: 165-168, 170-171, 173-174, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where the annotator does not report progress for cached tests, potentially causing confusion or errors.

Key Assertions:

- The `get_provider` method of `LLMAnnotator` returns a mock provider instance that is available.
- The `get_cache` method of `LLMCache` returns a mock cache instance with the expected annotation scenario.
- The `assemble` method of `ContextAssembler` returns a mock assembly result with the expected source and None value.
- The `append` method of `progress_msgs` appends a message indicating that the cached test is being annotated.
- Any messages in `progress_msgs` contain the string '(cache): test_cached'.
- The annotator reports progress for all cached tests, even if they are not actually annotated.

COVERAGE

src/pytest_llm_report/cache.py	1 lines (ranges: 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	37 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-84, 97-98, 100, 127, 129-135, 137, 139)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 by ensuring that the annotator does not attempt to annotate tests when the provider is unavailable.

Key Assertions:

- mocks.get_provider().is_available.return_value == False
- assert captured.out.startswith('not available. Skipping annotations')
- assert mock_provider.is_available.called_once_with() == True
- assert mock_provider.is_available.call_count == 1
- assert mock_provider.is_available.args[0] == 'ollama'
- assert mock_provider.return_value.message == 'not available. Skipping annotations'
- assert mock_provider.return_value.status_code == 500

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

📄 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 `extract_json_from_response` raises an error when encountering malformed JSON after extracting the response.

Why Needed: This test prevents a potential bug where the function `extract_json_from_response` fails to parse invalid JSON in the response, potentially leading to incorrect LLM output.

Key Assertions:

- The function `_parse_response` returns an error message indicating that the provided JSON is malformed.
- The function `_parse_response` raises a `JSONDecodeError` exception when encountering invalid JSON.
- The error message returned by `_parse_response` includes the string 'Failed to parse LLM response as JSON'.
- The test assertion checks for the presence of an error message in the output, indicating that the JSON is malformed.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	10 lines (ranges: 52-53, 186-187, 190-191, 194-195, 220-221)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Tests that the `base_parse_response` function handles non-string fields correctly when a list is expected.

Why Needed: This test prevents regression where the function incorrectly assumes a string value for non-string fields and returns an incorrect annotation.

Key Assertions:

- assert annotation.scenario == '123'
- assert annotation.why_needed == ['list']
- assert annotation.key_assertions == ['a']

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	22 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203-207, 212, 214-218)
`src/pytest_llm_report/llm/schemas.py`	7 lines (ranges: 38, 42-43, 50-53)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

📄 tests/test_base_maximal.py

9 tests

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

AI ASSESSMENT

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

Why Needed: Prevents a potential issue where the test relies on the `provider` being set to 'gemini' without ensuring it's actually configured correctly.

Key Assertions:

- The `config` object is created with the correct provider ('gemini').
- The `get_provider(config)` function returns an instance of `GeminiProvider`.
- The `provider` attribute of the returned `GeminiProvider` instance is set to 'gemini'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	10 lines (ranges: 52-53, 245, 247, 249, 252, 257, 262-263, 265)
src/pytest_llm_report/llm/gemini.py	7 lines (ranges: 134, 136-139, 141-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Tests the `get_invalid_provider` method when an unknown LLM provider is passed.

Why Needed: Prevents a potential ValueError that may occur if an invalid LLM provider is used in the configuration.

Key Assertions:

- The `Config` object passed to `get_provider` has an invalid provider.
- The error message returned by `get_provider` includes 'Unknown LLM provider: invalid'.
- The `pytest.raises` context manager ensures that a ValueError is raised with the specified match string.
- The `match` parameter in `pytest.raises` allows for a custom error message to be provided.
- The `Config` object passed to `get_provider` should not have an unknown LLM provider.
- The `get_provider` method should raise a ValueError when given 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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verifies that the `get_litellm_provider` function returns an instance of `LiteLLMProvider` when a valid configuration is provided.

Why Needed: Prevents a potential bug where the test fails if the configuration does not match any known provider.

Key Assertions:

- The returned value should be an instance of `LiteLLMProvider`.
- The returned value should have the correct class name (`LiteLLMProvider`).
- The returned value should have a valid `__init__` method that accepts a `Config` object as an argument.
- The `get_provider` function should return an instance of `LiteLLMProvider` when given a valid configuration.
- The test should pass if the configuration is valid and the provider is created successfully.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	9 lines (ranges: 52-53, 245, 247, 249, 252, 257-258, 260)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the `get_noop_provider` function might return an incorrect or unexpected result if no provider is provided.

Key Assertions:

- The returned value should be an instance of `NoopProvider`.
- The `provider` attribute of the returned value should not be `None`.
- The `isinstance()` method should correctly identify the type of the returned value as `NoopProvider`.
- The `get_provider()` function should be able to successfully retrieve a provider instance without any issues.
- Any exceptions raised by the `get_provider()` function should not affect the correctness of this test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 245, 247, 249-250)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify the correctness of getting an Ollama provider from a config object.

Why Needed: This test prevents regression where the `get_ollama_provider` function returns an incorrect type (e.g., not OllamaProvider) when given a valid config.

Key Assertions:

- The returned value is indeed of type OllamaProvider.
- The provider instance has the correct class attribute 'provider' set to 'ollama'.
- The provider instance has the correct method 'get_ollama_provider()' implemented.
- The provider instance does not have any other attributes or methods that are not required for an OllamaProvider instance.
- The provider instance is not None and has a valid configuration object.
- The provider instance is of the correct class (OllamaProvider).
- The provider instance's 'provider' attribute matches the expected value ('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: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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 whether caches are available.

Why Needed: This test prevents regression in cases where the LLM provider does not implement _check_availability.

Key Assertions:

- The `is_available()` method should return True for both instances of the provider.
- The `is_available()` method should return True when called multiple times.
- The `checks` attribute 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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The `get_model_name()` method of the `ConcreteProvider` class should return the default model name specified in the configuration.

Why Needed: This test prevents a potential bug where the default model name is not correctly retrieved from the configuration, potentially leading to unexpected behavior or errors.

Key Assertions:

- The `get_model_name()` method of the `ConcreteProvider` class should return 'test-model' when called with a `Config` object that has a `model` attribute set to 'test-model'.
- The `get_model_name()` method of the `ConcreteProvider` class should raise an error if the configuration does not specify a model name.
- The `get_model_name()` method of the `ConcreteProvider` class should return the default model name specified in the configuration when called with a valid `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/llm/base.py	3 lines (ranges: 52-53, 136)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the get_rate_limits method returns None when no rate limits are specified in the configuration.

Why Needed: This test prevents a potential bug where the default rate limit of None is returned unexpectedly without proper justification or context.

Key Assertions:

- The provider instance does not have any rate limits set.
- The get_rate_limits method returns None for all configurations with no specified rate limits.
- The configuration object passed to the provider has a valid default value for rate limits (None).
- The provider instance is created without specifying any rate limits, and the get_rate_limits method does not raise an exception or return an error.
- The get_rate_limits method returns None when called on a configuration with no specified rate limits.
- The rate limit values are correctly set to None for all configurations with no specified rate limits.
- The provider instance is created without specifying any rate limits, and the get_rate_limits method does not raise an exception or return an error.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 128)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `provider` object is not local to the test environment.
- The `provider.is_local()` method returns False.
- A non-local configuration is used with the `ConcreteProvider` instance.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	3 lines (ranges: 52-53, 147)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

📄 tests/test_cache.py

7 tests

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

AI ASSESSMENT

Scenario: Verify that the hash of a function with the same source code is consistent.

Why Needed: This test prevents regression when the source code changes and the hash should still match.

Key Assertions:

- The hash of `source` should be equal to itself.
- The hash of `source` should not change even if `source` is modified.
- If `source` is copied, its hash should remain unchanged.

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: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `hash_source` function should return a different hash value for two different source strings.
- Two different source strings should have different hashes.
- The `hash_source` function should not be able to find a common prefix between two source strings and produce the same hash.
- The `hash_source` function should raise an error when given a source string that is empty or does not contain any code.
- The `hash_source` function should return the correct hash value for a source string containing only whitespace characters.
- Two different source strings with the same prefix but different suffixes should have different hashes.
- The `hash_source` function should raise an error when given a source string that contains invalid Python syntax.

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: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The hash should be exactly 16 characters long.
- The hash should not have any leading zeros.
- The hash should not have trailing zeros.
- The hash should only contain hexadecimal digits (0-9, A-F, a-f).
- No whitespace or special characters should be present in the hash.
- The hash should be a valid SHA-256 hash.
- The hash should not be empty.

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: 380-381, 384, 388-390)`

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

AI ASSESSMENT

Scenario: Test the `clear` method of LlmCache to ensure it properly removes all cache entries.

Why Needed: This test prevents a potential bug where cache entries are not correctly removed when the `clear` method is called.

Key Assertions:

- Verify that the `clear` method returns the correct number of cache entries (2 in this case).
- Verify that no cache entries are retrieved from the cache after calling `clear`.
- Check if the cache entry for 'test::a' with hash 'hash1' is successfully removed.
- Check if the cache entry for 'test::b' with hash 'hash2' is successfully removed.
- Verify that the cache directory remains empty after clearing all entries.

COVERAGE

src/pytest_llm_report/cache.py	26 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 129, 132-136, 141)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that annotations with errors are not cached.

Why Needed: To prevent caching of error-related annotations, making it easier to test for errors without having to manually delete the cache.

Key Assertions:

- The annotation 'error' is present in the cache.
- The value of the annotation 'error' matches the expected string 'API timeout'.
- The annotation 'error' is not present in the cache after calling get() on it.
- The result of getting the annotation 'error' from the cache does not match the expected string 'API timeout'.
- The annotation 'error' has a different value than the expected string 'API timeout' in the cache.
- The cache is empty before calling get(), but the annotation 'error' is present in it after calling get().

COVERAGE

`src/pytest_llm_report/cache.py`	11 lines (ranges: 39-41, 53, 55-56, 86, 88, 118-119, 121)
`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify that the 'get_missing' test returns None for missing entries in the cache.

Why Needed: This test prevents a potential bug where the 'get' method of LlmCache returns None when an entry does not exist in the cache.

Key Assertions:

- The function `cache.get('test::foo', 'abc123')` should return `None` for missing entries.
- The cache should be able to store and retrieve data without causing a KeyError.
- The test should fail when trying to access an entry that does not exist in the cache.
- The `get` method of LlmCache should raise a KeyError when called with a non-existent key.
- The error message should indicate that the requested key does not exist in the cache.
- The cache should be able to handle missing entries without causing a crash or unexpected behavior.

COVERAGE

src/pytest_llm_report/cache.py	9 lines (ranges: 39-41, 53, 55-56, 118-119, 121)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test the ability to store and retrieve annotations from the cache.

Why Needed: This test prevents a potential bypass attack by ensuring that the cache stores and retrieves annotations in a controlled manner.

Key Assertions:

- Verify that the annotation is stored correctly in the cache with the given key.
- Check that the retrieved annotation matches the expected scenario and confidence level.
- Ensure that the cache does not return None 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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

📄 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 node id and message.

Why Needed: This test prevents a bug where the structure of collection errors is not correctly identified, potentially leading to incorrect handling or reporting of these errors in the application.

Key Assertions:

- The error object has the correct 'nodeid' attribute set to 'test_bad.py'.
- The error object has the correct 'message' attribute set to 'SyntaxError'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where an error is thrown when trying to collect errors from an empty collection, potentially causing test failures or unexpected behavior.

Key Assertions:

- The `get_collection_errors()` method returns an empty list.
- No exception is raised when calling `get_collection_errors` on an empty configuration.
- A valid `TestCollector` instance with an initially empty configuration can be created and used without errors.

COVERAGE

`src/pytest_llm_report/collector.py`	15 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 285)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the default llm_context_override might not be correctly set in some cases, potentially leading to incorrect results or unexpected behavior.

Key Assertions:

- The llm_context_override attribute of TestCaseResult node should be None.
- The llm_context_override attribute is correctly set to None for all TestCaseResult nodes.
- If the default llm_context_override is not set, pytest_llm_report will raise an AssertionError with a meaningful message.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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

AI ASSESSMENT

Scenario: Test the default value of llm_opt_out for LLMOptOutDefaultFalse.

Why Needed: Prevents regression in LLMOptOutDefaultFalse test case when llm_opt_out is set to True by default.

Key Assertions:

- The llm_opt_out attribute should be False.
- The TestCaseResult instance should have an llm_opt_out attribute of False.
- The TestCaseResult instance's llm_opt_out attribute should not be set to True.
- If llm_opt_out is set to True, the test should fail with a meaningful error message.
- The test should verify that the TestCaseResult instance has been created successfully.
- The TestCaseResult instance should have a valid nodeid and outcome.
- The TestCaseResult instance's llm_opt_out attribute should be False after calling the test_llm_opt_out_default_false function.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The `capture` option is disabled by default for the `CollectorOutputCapture` class.

Why Needed: This test prevents a bug where the output capture feature is not enabled by default, potentially leading to unexpected behavior or errors.

Key Assertions:

- assert config.capture_failed_output is False
- assert isinstance(config, Config)
- assert hasattr(config, 'capture') and config.capture is False
- assert config.capture is None

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the default value of `capture_output_max_chars` in the `Config` class is 4000.

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

Key Assertions:

- The value of `capture_output_max_chars` in the `Config` class should be 4000.
- The maximum number of characters that can be captured by `capture_output_max_chars` is 4000.
- Setting `capture_output_max_chars` to a non-default value could lead to unexpected behavior or errors when capturing output.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test 'xfail failures should be recorded as xfailed' verifies that TestCollector handles xfail failures correctly.

Why Needed: This test prevents regression where TestCollector incorrectly records failed tests as expected failures.

Key Assertions:

- The `results` dictionary contains the correct key for the failed test, which is 'xfailed'.
- The `outcome` attribute of the failed test result is set to 'xfailed' instead of 'expected failure'.
- The `wasxfail` attribute of the failed test result is set to 'expected failure' instead of 'xfail'.

COVERAGE

`src/pytest_llm_report/collector.py`	`36 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-118, 124, 127, 140, 155-159, 163, 167, 171, 209-210, 212, 216, 227-228, 230-234, 238)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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

AI ASSESSMENT

Scenario: The test verifies that xfail passes are correctly recorded as xpassed in the report.

Why Needed: This test prevents regression where an xfail might pass due to a temporary error or unexpected behavior.

Key Assertions:

- The `result.outcome` is set to 'xpassed' after handling the runtest logreport.
- The `collector.results[report.nodeid]` returns an object with `outcome` attribute equal to 'xpassed'.
- The `report.wasxfail` is not present in the result, indicating xfail passes are correctly recorded as xpassed.

COVERAGE

src/pytest_llm_report/collector.py	26 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-115, 124, 127, 140, 155-159, 163, 167, 171, 209-210, 212-214)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test the creation of a Collector instance with an empty configuration.

Why Needed: This test prevents a potential issue where the Collector is created without any initial results, potentially causing incorrect assertions in subsequent tests.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the `get_results` method returns a sorted list of node IDs.

Why Needed: This test prevents regression where the order of results is not preserved due to manual sorting.

Key Assertions:

- The list of node IDs returned by `get_results()` should be in ascending order.
- The list of node IDs returned by `get_results()` should contain only unique node IDs.
- No duplicate node IDs should appear in the list.
- All node IDs should be present in the list.
- Node IDs should be sorted alphabetically (case-insensitive).
- Node IDs should be sorted numerically (0-based).

COVERAGE

`src/pytest_llm_report/collector.py`	15 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the count of collected items is not updated correctly when an item is deselected.

Key Assertions:

- The `collected_count` attribute should be set to 3 after simulating 3 collected items.
- The `deselected_count` attribute should be set to 1 after simulating 1 deselected item.
- The `collected_count` and `deselected_count` attributes should match the expected values before calling `handle_collection_finish`.

COVERAGE

`src/pytest_llm_report/collector.py`	16 lines (ranges: 78-79, 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

📄 tests/test_collector_maximal.py

14 tests

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

AI ASSESSMENT

Scenario: Test that `capture_output` is disabled when `config.capture_failed_output=False` and the test is run via handle_runtest_logreport.

Why Needed: This test prevents a regression where the collector captures output even if `capture_failed_output` is set to False, which could lead to unexpected behavior or errors in downstream tests.

Key Assertions:

- The `collector.handle_runtest_logreport(report)` call should not capture any output.
- The `results` dictionary of the test 't' should have a `captured_stdout` key with `None` as its value.
- The `wasxfail` attribute of the report should be deleted after processing.
- The `collector.results['t']` variable should still contain the original result before it was modified by the collector.

COVERAGE

src/pytest_llm_report/collector.py	36 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-118, 124, 127-128, 130, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `captured_stderr` attribute of the `TestCaseResult` object should be set to 'Some error'.
- The `report.capstderr` method should have been called with a string argument equal to 'Some error'.
- The `collector._capture_output(result, report)` function should have captured stderr and stored it in the `captured_stderr` attribute of the `TestCaseResult` object.

COVERAGE

`src/pytest_llm_report/collector.py`	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264, 268-269)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the `test_capture_output_stdout` function captures stdout correctly.

Why Needed: This test prevents a potential regression where the collector fails to capture stdout.

Key Assertions:

- The captured stdout is set to 'Some output'.
- The captured stderr is set to an empty string.
- The `test_capture_output_stdout` function is called with a `report` object that has both `capstdout` and `capstderr` attributes set.

COVERAGE

src/pytest_llm_report/collector.py	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264-265, 268)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the collector truncates output exceeding max chars when capture_failed_output is True.

Why Needed: This test prevents a potential bug where the captured output exceeds the maximum allowed characters, causing unexpected behavior or errors.

Key Assertions:

- The captured stdout should be truncated to 10 characters.
- The captured stderr should not have any characters.
- The captured stdout is exactly 9 characters long (1234567890).
- The captured stderr is empty.

COVERAGE

`src/pytest_llm_report/collector.py`	18 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 261, 264-265, 268)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression in cases where an item is marked as 'llm_opt_out' or has overridden 'llm_context_override'.

Key Assertions:

- item.callspec.id should be set to 'param1'
- result.param_id should be set to 'param1'
- result.llm_opt_out should be True
- result.llm_context_override should be set to 'complete'
- result.requirements should contain ['REQ-1', 'REQ-2']

COVERAGE

src/pytest_llm_report/collector.py	35 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 155-159, 163-164, 167-169, 171, 181-182, 185-189, 198-200, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the `test_extract_error_repr_crash` function handles ReprFileLocation correctly.

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

Key Assertions:

- The `_extract_error` method of `TestCollector` should return the expected error message when `report.longrepr.__str__.return_value = 'Crash report'`.
- When `report.longrepr.__str__.return_value is not 'Crash report'`, the function should still work as expected.
- The `_extract_error` method of `TestCollector` should handle cases where `report.longrepr.__str__.return_value` is a string that cannot be converted to an error message.
- When `report.longrepr.__str__.return_value is not a string`, the function should raise an exception with a meaningful error message.
- The `_extract_error` method of `TestCollector` should correctly handle cases where `report.longrepr.__str__.return_value is None` or an empty string.

COVERAGE

src/pytest_llm_report/collector.py	22 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 227-228, 230-234, 238)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the `_extract_error` method returns a string representing an error message.

Why Needed: Prevents a potential bug where the extracted error message is not a valid representation of an error.

Key Assertions:

- The `report.longrepr` attribute should be set to a string value.
- The `collector._extract_error(report)` method should return a string that matches the expected error message.
- The error message returned by `_extract_error` should not be empty or None.
- The error message returned by `_extract_error` should contain the word 'error'.
- The error message returned by `_extract_error` should not contain any leading or trailing whitespace.
- The error message returned by `_extract_error` should not contain any special characters other than those allowed in Python strings.
- The error message returned by `_extract_error` should be a string that can be concatenated with the `longrepr` attribute without causing an 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: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test the `_extract_skip_reason` method with a `report` that does not contain longrepr.

Why Needed: To prevent a potential issue where the test fails if no longrepr is present in the report.

Key Assertions:

- The method should return `None` when `report.longrepr` is `None`.
- The method should not raise an exception or throw an error when `report.longrepr` is `None`.
- The method should correctly handle cases where `report.longrepr` is `None` without raising an exception.
- The `_extract_skip_reason` method should be able to extract the reason for skipping from a report with no longrepr.
- A test case with a report that does not contain longrepr should pass this test.
- A test case with a report that contains longrepr but is still marked as `None` (i.e., it's an empty string) should also pass this test.
- The method should correctly handle the case where `report.longrepr` is an empty string (`''`).

COVERAGE

`src/pytest_llm_report/collector.py`	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250, 252)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test `test_extract_skip_reason_string` verifies that the `_extract_skip_reason` method returns a string 'Just skipped' as skip reason.

Why Needed: This test prevents a potential bug where the `longrepr` attribute of a report object is not correctly set to the expected value 'Just skipped'.

Key Assertions:

- The `report.longrepr` attribute should be set to 'Just skipped'.
- The `assert` statement will raise an AssertionError if `report.longrepr` is not equal to 'Just skipped'.
- The `_extract_skip_reason` method correctly extracts the string value from the report object.
- The expected value of `report.longrepr` matches the actual value returned by `_extract_skip_reason`.
- If a different skip reason is used instead of 'Just skipped', the test will fail with an AssertionError.

COVERAGE

`src/pytest_llm_report/collector.py`	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250-251)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the `extract_skip_reason_tuple` function correctly extracts skip messages from tuples containing a longrepr.

Why Needed: This test prevents a potential regression where the `extract_skip_reason_tuple` function fails to extract skip messages from tuples with longreprs.

Key Assertions:

- The tuple passed to `_extract_skip_reason` should contain all required elements: `(file, line, message)`.
- The `longrepr` attribute of the report object should be a string containing the file name and line number and 'Skipped for reason' as its value.
- The `str(report.longrepr)` expression should return a string that includes the longrepr tuple.
- The `in collector._extract_skip_reason(report)` assertion should pass if the `longrepr` attribute of the report object is present in the `_extract_skip_reason` function's output.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 250-251)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: When the `handle_collection_report` method is called with a report containing an error, it should record this error in the collection.

Why Needed: This test prevents a potential bug where the collector does not handle reports containing errors correctly and fails to record them in the collection.

Key Assertions:

- The `collection_errors` list contains exactly one element with the specified nodeid.
- The first element of `collection_errors` has the correct message 'SyntaxError'.
- All other elements in `collection_errors` have a valid nodeid and message.

COVERAGE

src/pytest_llm_report/collector.py	21 lines (ranges: 58, 60-65, 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- res.rerun_count should be equal to 1 (expected)
- res.final_outcome should be 'failed' (expected)
- collector.results['t::r'] should contain the expected data (expected)

COVERAGE

`src/pytest_llm_report/collector.py`	36 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-118, 124, 127-128, 130, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression in case of a failure during runtest logreport, which could lead to incorrect or missing results being reported.

Key Assertions:

- res.outcome == 'error'
- res.phase == 'setup'
- res.error_message == 'Setup failed'

COVERAGE

`src/pytest_llm_report/collector.py`	36 lines (ranges: 90, 93-94, 96, 99-103, 109-112, 114-115, 124, 127, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where a teardown failure is not properly handled and instead, it logs an error with a meaningful message.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	`38 lines (ranges: 90, 93-94, 96, 99, 110-112, 114-115, 124, 127-128, 130, 132-133, 135-137, 140, 155-159, 163, 167, 171, 209-210, 227-228, 230-234, 238)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

📄 tests/test_coverage_boosters.py

3 tests

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

AI ASSESSMENT

Scenario: Verifies that the `GeminiProvider` class correctly handles edge cases in model parsing.

Why Needed: This test prevents regression in case a new gemini model is added and the parser does not handle it properly.

Key Assertions:

- The function `_parse_preferred_models()` returns an empty list when `None` as the model configuration.
- The function `_parse_preferred_models()` returns an empty list when 'all' is set as the model configuration.
- The function `_parse_preferred_models()` correctly handles models with no configuration by returning 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	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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 potential bug where the rate limiter allows too many tokens to be recorded before reaching the limit, potentially causing unexpected behavior or errors.

Key Assertions:

- assert limiter.next_available_in(60) > 0
- assert limiter.next_available_in(10) == 0
- assert limiter.next_available_in(100) < 0

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/gemini.py`	35 lines (ranges: 39-42, 45-46, 48, 52-54, 66, 68-70, 81-82, 84, 87-88, 92-93, 95-96, 100-101, 103, 105, 107-114)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify that the `to_dict()` method of SourceCoverageEntry and LlmAnnotation correctly converts them into dictionaries with expected values.

Why Needed: This test prevents regression in coverage booster functionality, ensuring accurate conversion of source code coverage entries and annotations to dictionary formats.

Key Assertions:

- The 'coverage_percent' attribute of the `SourceCoverageEntry` object is set to 50.0 as expected.
- The 'error' attribute of the `LlmAnnotation` object is set to 'timeout' as expected.
- The 'duration' attribute of the `RunMeta` object is set to 1.0 as expected.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: 380-381, 384, 388-390)

📄 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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `get_warnings` method returns an incorrect or empty list of warnings, potentially masking real issues with the coverage data.

Key Assertions:

- The function `get_warnings()` should return a list of warnings.
- The function `get_warnings()` should not be null or undefined.
- The function `get_warnings()` should return at least one warning if there are any warnings in the coverage data.
- The function `get_warnings()` should raise an AssertionError if it returns an incorrect or empty list of warnings.
- The function `get_warnings()` should handle cases where the coverage data is null or undefined without raising an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	3 lines (ranges: 44-45, 308)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function should return an empty dictionary even if no coverage file exists.
- The function should not have any warnings in this case.
- The `map_coverage` method should be able to handle a mock Path.exists and glob.glob that returns empty lists.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `_extract_nodeid()` method of the `CoverageMapper` returns the expected node ID for each phase.
- The node IDs returned by `_extract_nodeid()` are consistent across different test files and phases.
- The coverage map is complete when all phases have been included.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	11 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the `extract_nodeid_filters_setup` method of `CoverageMapper` returns `None` when the `include_phase` parameter is set to `'run'`.

Why Needed: This test prevents a potential regression where the `nodeid` filter might be incorrectly applied during coverage extraction, leading to incorrect results.

Key Assertions:

- The method `_extract_nodeid` of `CoverageMapper` should return `None` when the input string is in the format `test.py::test_foo|setup`.
- The method `_extract_nodeid` of `CoverageMapper` should not return any node IDs when the `include_phase` parameter is set to `'run'`.
- The test should fail with an assertion error if the `nodeid` filter does not match any nodes in the code.
- The test should verify that the `nodeid` filter correctly excludes setup phase nodes from coverage extraction.
- The method `_extract_nodeid` of `CoverageMapper` should return a node ID or None when the input string is in the correct format and the `include_phase` parameter is set to `'run'`.
- The test should verify that the `nodeid` filter correctly handles cases where the input string contains multiple nodes separated by `|`.
- The method `_extract_nodeid` of `CoverageMapper` should raise an assertion error if the input string does not match any node IDs or is not in the correct format when the `include_phase` parameter is set to `'run'`.
- The test should verify that the `nodeid` filter correctly handles cases where the input string contains nodes with different names separated by `|`.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/coverage_map.py`	9 lines (ranges: 44-45, 216, 220, 224-225, 228-230)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify that the `extract_nodeid_with_run_phase` test extracts the correct node ID from the run phase context.

Why Needed: This test prevents a potential bug where the node ID is not extracted correctly when running in the run phase.

Key Assertions:

- The function `_extract_nodeid` should be able to extract the `node_id` from the given string and return it.
- The returned value of `_extract_nodeid` should match the expected `node_id` value.
- The test should fail if the extracted node ID does not match 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/coverage_map.py	11 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231, 233, 236)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

📄 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 test_extract_contexts_full_logic function exercises all paths in _extract_contexts by verifying coverage of app.py files.

Why Needed: This test prevents regression where the function does not cover all possible paths in the codebase, potentially leading to incorrect or incomplete results.

Key Assertions:

- The function should return a list containing 'test_app.py::test_one' and 'test_app.py::test_two'.
- The function should verify that app.py is included in test_one's coverage.
- The function should correctly count the number of lines in each context for app.py.
- The function should not include any other files than app.py in its return value.
- The function should handle cases where mock_data.contexts_by_lineno returns an empty dictionary.
- The function should raise an AssertionError if it does not find 'test_app.py::test_one' or 'test_app.py::test_two' in the result.
- The function should correctly exclude contexts that are not actually covered by app.py files.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/coverage_map.py`	`57 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 137-140, 144, 148, 150, 152-153, 156, 160-163, 165, 167-168, 173, 176, 178-184, 187-189, 191-194, 196, 199-200, 202, 216, 220, 224-225, 228-229, 231, 233, 236)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`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 of CoverageMapper correctly handles data with no test contexts.

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

Key Assertions:

- mocked return value of mock_data.contexts_by_lineno should be an empty dictionary.
- mocked return value of mock_data.measured_files should match the input data.
- result should be an empty dictionary after calling _extract_contexts on mock_data.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/coverage_map.py`	14 lines (ranges: 44-45, 118, 121-122, 127, 131-135, 144-146)
`src/pytest_llm_report/errors.py`	4 lines (ranges: 139-142)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression in coverage analysis when using a context without a phase.

Key Assertions:

- The `_extract_nodeid` method should return the expected node ID for each line in the given context.
- The `None` value should be returned for lines that are filtered out by the `CoverageMapper` class.
- The `_extract_nodeid` method should ignore nodes without a phase when extracting variants.
- The test should pass even if the `test.py::test_no_phase` context is used, as it does not contain any lines to extract.
- The `None` value for `test.py::test|run` line in the `teardown` phase should be ignored by the `CoverageMapper` class.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	14 lines (ranges: 44-45, 216, 220, 224-225, 228-229, 231-234, 236, 239)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the function does not warn users about missing coverage data.

Key Assertions:

- The function should return None for the coverage data.
- The warning message should be 'W001'.
- There should be exactly one warning.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	9 lines (ranges: 44-45, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the test_load_coverage_data_read_error function prevents a regression by handling errors reading coverage files.

Why Needed: This test prevents a potential regression where the CoverageMapper class fails to handle corrupted coverage files, potentially leading to unexpected behavior or errors in subsequent tests.

Key Assertions:

- The function should return None when an error occurs while loading coverage data.
- Any warnings generated by the mapper should contain the message 'Failed to read coverage data'.
- The mapper's warnings should not contain any other messages.
- The function should mock the CoverageData class with a MagicMock instance that raises an Exception on read.
- The mocked CoverageData instance should have a read method that returns an Exception.
- The _load_coverage_data method of the CoverageMapper class should return None when an error occurs.
- Any exceptions raised by the mapper's warnings should be caught and ignored.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/coverage_map.py`	17 lines (ranges: 44-45, 72-73, 83, 86, 88, 92, 94-96, 107-111, 114)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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 CoverageData instances.

Why Needed: This test prevents regression where the CoverageMapper fails to update the CoverageData instances for parallel coverage files.

Key Assertions:

- The mock instances of CoverageData returned by the CoverageData class should have been updated at least twice.
- The update method of CoverageData instance should be called at least twice during the test execution.
- The mock instances of CoverageData should not be None after calling the _load_coverage_data() method.
- The mock instances of CoverageData should have different values before and after calling the _load_coverage_data() method.
- The update method of CoverageData instance should be called at least twice during the test execution.
- The mock instances of CoverageData should not be None after calling the _load_coverage_data() method.
- The mock instances of CoverageData should have different values before and after calling the _load_coverage_data() method.
- The update method of CoverageData instance should be called at least twice during the test execution.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/coverage_map.py`	`15 lines (ranges: 44-45, 72-73, 83, 86, 88, 92, 94, 98, 101-104, 106)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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

AI ASSESSMENT

Scenario: Test should handle case where _load_coverage_data returns None.

Why Needed: Prevents regression in coverage mapping functionality when no data is provided.

Key Assertions:

- The function mapper._load_coverage_data() returns None when config._load_coverage_data() returns None.
- The result of the map_coverage() method should be an empty dict when _load_coverage_data() returns None.
- No exception should be raised when _load_coverage_data() returns None.
- The test should pass even if mapper._load_coverage_data() returns None, as it's a valid scenario.
- Other potential exceptions (e.g., ValueError) should not be raised in this case.
- The function mapper.map_coverage() should return an empty dict when _load_coverage_data() returns None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	5 lines (ranges: 44-45, 58-60)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the CoverageMapperMaximal class correctly skips files with errors during analysis.

Why Needed: This test prevents a potential regression where an error during coverage analysis would incorrectly include files in the coverage report.

Key Assertions:

- mock_cov.analysis2.assert_called_once_with(mock_data)
- mock_cov.get_data.return_value measured_files().return_value == ['app.py']
- entries is empty after skipping files with errors

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/coverage_map.py	22 lines (ranges: 44-45, 243-244, 246-248, 250, 252-254, 259, 261, 263-268, 271, 299-300)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 mapper.map_source_coverage with comprehensive coverage of all paths in map_source_coverage.

Why Needed: Prevents regression where only specific files are covered by the mapper, potentially leading to incomplete or inaccurate coverage reports.

Key Assertions:

- The function mapper.map_source_coverage returns exactly one entry with file path 'app.py', which is covered by 66.67% of statements.
- The entry has three statements and two lines that were not covered (covered = 2, missed = 1).
- The coverage percentage is 66.67%, indicating comprehensive coverage of all paths in map_source_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`	32 lines (ranges: 44-45, 243-244, 246-248, 250, 252, 259-261, 273, 276-279, 281-283, 285-293, 295, 299-300)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`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 identifies and returns a WarningCode.W001_NO_COVERAGE error.

Why Needed: To prevent a regression where the test fails when an unknown code is passed to the make_warning function.

Key Assertions:

- The function should return a WarningCode.W001_NO_COVERAGE with the given message.
- The function should set the detail attribute of the warning object to 'test-detail'.
- The function should not raise any exceptions when an unknown code is passed.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/errors.py`	4 lines (ranges: 139-142)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that warning codes have correct values.

Why Needed: Prevents a potential bug where the warning code values are incorrect, potentially leading to unexpected behavior or errors in the application.

Key Assertions:

- {'message': 'assert WarningCode.W001_NO_COVERAGE.value == "W001"', 'expected': 'W001'}
- {'message': 'assert WarningCode.W101_LLM_ENABLED.value == "W101"', 'expected': 'W101'}
- {'message': 'assert WarningCode.W201_OUTPUT_PATH_INVALID.value == "W201"', 'expected': 'W201'}
- {'message': 'assert WarningCode.W301_INVALID_CONFIG.value == "W301"', 'expected': 'W301'}
- {'message': 'assert WarningCode.W401_AGGREGATE_DIR_MISSING.value == "W401"', 'expected': '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: 380-381, 384, 388-390)

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 creating a new Warning instance.
- The `message` attribute of the Warning object should be set to 'No coverage' when creating a new Warning instance without providing a detail message.
- The `detail` attribute of the Warning object should be provided when creating a new Warning instance with a detail message.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/errors.py`	`6 lines (ranges: 70-72, 74-76)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

📄 tests/test_errors_maximal.py

6 tests

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where a warning is not created for known code, potentially causing issues in downstream code.

Key Assertions:

- The function `make_warning` returns an instance of `WarningCode.W101_LLM_ENABLED` with the correct code.
- The message returned by `make_warning` matches the expected value from `WARNING_MESSAGES[WarningCode.W101_LLM_ENABLED]`.
- The detail attribute is set to None, as it should be for known code 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/errors.py`	`4 lines (ranges: 139-142)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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

AI ASSESSMENT

Scenario: Test MakeWarning::test_make_warning_unknown_code verifies that the test uses a fallback message for unknown code when an enum is not allowed.

Why Needed: This test prevents a bug where the typed function would raise a TypeError if it tries to access a non-enum value, but instead provides a meaningful fallback message.

Key Assertions:

- The assertion `w.message == 'Unknown warning.'` checks that the returned warning message is indeed 'Unknown warning.'
- The assert statement `try: w = make_warning(missing_code)` verifies that an exception is raised when trying to create a warning for unknown code.
- The finally block `finally: WARNING_MESSAGES[missing_code] = old_message` restores the original message 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/errors.py`	4 lines (ranges: 139-142)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where a warning is not correctly created when a detail message is provided.

Key Assertions:

- w.code == WarningCode.W301_INVALID_CONFIG
- w.detail == 'Bad value'
- assert w.detail in ['Bad value', 'Invalid configuration']
- w.detail != 'Unknown error'
- w.detail != 'Configuration is valid'
- w.detail != 'Value is invalid'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Testing the `test_codes_are_strings` function to ensure enum values are strings.

Why Needed: This test prevents a potential warning that occurs when trying to use non-string enum values.

Key Assertions:

- assert isinstance(code.value, str) checks if each enum value is indeed a string.
- assert code.value.startswith('W') checks if the first letter of each string value starts with 'W'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 issue where the warning is not properly serialized to a dictionary with detailed information.

Key Assertions:

- The warning object `w` should be converted into a dictionary using its `to_dict()` method.
- The resulting dictionary should contain only the required keys: `code` and `message`.
- The value of `code` in the dictionary should match the expected value 'W001'.
- The value of `message` in the dictionary should match the expected value 'No coverage'.
- The warning object `w` should not contain any additional keys or values.
- The warning object `w` should be a valid instance of the Warning class.
- The warning object `w` has been properly initialized with the required attributes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	5 lines (ranges: 70-72, 74, 76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test warning serialization with detailed message.

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

Key Assertions:

- The 'detail' key in the dictionary should contain the 'Check setup' message.
- The 'code' key in the dictionary should be set to 'W001'.
- The 'message' key in the dictionary should be set to 'No coverage'.
- The 'data' attribute of the Warning object should have a dictionary with the required keys and values.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/errors.py	6 lines (ranges: 70-72, 74-76)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

📄 tests/test_fs.py

12 tests

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

AI ASSESSMENT

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

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

Key Assertions:

- assert is_python_file('foo/bar.txt') is False
- assert is_python_file('foo/bar.pyc') is False

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	1 lines (ranges: 79)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function should return `True` when given a path to a `.py` file.
- The function should handle paths with different extensions (e.g., `.java`, `.cpp`) correctly.
- The function should not incorrectly identify non-`.py` files as Python files (e.g., `.txt`, `.js`).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	1 lines (ranges: 79)

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

AI ASSESSMENT

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

Why Needed: This test prevents regression where the function does not correctly handle paths relative to the test directory.

Key Assertions:

- The file is created with the correct path (subdir/subdir/file.py).
- The parent directory of the file exists and is created if it does not exist.
- The file is successfully touched after being created.
- The result of making the relative path absolute matches the expected output ('subdir/file.py').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	11 lines (ranges: 30, 33, 36, 39, 42, 55, 58-60, 63-64)

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

AI ASSESSMENT

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

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

Key Assertions:

- The result of `make_relative('foo/bar')` should be 'foo/bar' because there are no relative paths.
- The current implementation does not handle cases where the input path has no base.
- The function should raise an error or return a specific value when no base is provided.
- The normalized path should always start with the root directory.
- The function should handle absolute paths correctly.
- The test should be able to reproduce the issue and provide a clear explanation of what went wrong.
- The current implementation does not support relative paths starting with a dot (..).
- The test should verify that the function returns an empty string for an empty input path.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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

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: 380-381, 384, 388-390)
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 `normalize_path` function should correctly handle forward slashes in file paths.

Why Needed: This test prevents a potential bug where the function does not properly handle forward slashes in file paths, potentially leading to incorrect results or errors.

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: 380-381, 384, 388-390)
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

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: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	5 lines (ranges: 30, 33, 36, 39, 42)

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

AI ASSESSMENT

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

Why Needed: This test prevents the inclusion of paths that match custom exclusion patterns, ensuring the test suite remains focused on expected files.

Key Assertions:

- The function `should_skip_path` returns True for 'tests/conftest.py' with exclude_patterns=['test*']
- The function `should_skip_path` returns False for 'src/module.py' with exclude_patterns=['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: 380-381, 384, 388-390)
src/pytest_llm_report/util/fs.py	15 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116-117, 119-121, 123)

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

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the function `should_skip_path` incorrectly skips normal file system paths.

Key Assertions:

- assert should_skip_path('src/module.py') is False
- assert should_skip_path('src/tests/test_module.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: 380-381, 384, 388-390)
`src/pytest_llm_report/util/fs.py`	11 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-112, 116, 123)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential issue where the function incorrectly skips non-existent Git repositories.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`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: 380-381, 384, 388-390)
`src/pytest_llm_report/util/fs.py`	10 lines (ranges: 30, 33, 36, 39, 42, 100, 103, 111-113)

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

AI ASSESSMENT

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

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

Key Assertions:

- The function should return True for venv directories in the 'lib/python/site.py' directory, and False otherwise.
- The function should return True for .venv directories in the 'lib/python/site.py' directory, and False otherwise.
- The function should correctly handle other venv directories as well.
- The function should not incorrectly identify non-existent or system-wide venv directories as being to be skipped.
- The function should work correctly with Python versions that use different paths for site-packages (e.g. 3.x vs 2.x).

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`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: Test the effect of pruning on request and token usage counts.

Why Needed: This test prevents a regression where requests are not properly cleared from the rate limiter after a long time.

Key Assertions:

- The length of _request_times should be zero.
- _request_times should contain only one element (the last added request).
- The length of _token_usage should be zero.
- _token_usage should contain only one element (the last added token usage).

COVERAGE

src/pytest_llm_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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify that the rate limiter prevents requests from exceeding a certain threshold (in this case, 1 request per minute) within a specified time frame (1 minute).

Why Needed: This test prevents a potential issue where a request is allowed to exceed the rate limit and then becomes unavailable due to being blocked by the rate limiter.

Key Assertions:

- The `next_available_in` method should return a value greater than 0 within the specified time frame (1 minute).
- The `next_available_in` method should return a value less than or equal to 60.0 minutes after the request was recorded.

COVERAGE

`src/pytest_llm_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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify that the rate limiter allows 90 tokens to pass before waiting for 20 more tokens.

Why Needed: This test prevents a potential bug where the rate limiter does not allow enough tokens to pass before waiting for more.

Key Assertions:

- The number of available tokens in the rate limiter should be greater than 0 after recording 90 tokens.
- The number of available tokens in the rate limiter should be equal to 2 after recording an additional 10 tokens.
- The time elapsed since the last token was recorded should be less than or equal to 20 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`	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: 380-381, 384, 388-390)

PASSED `tests/test_gemini_advanced.py::TestGeminiRateLimiter::test_wait_for_slot` 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`	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: 380-381, 384, 388-390)

📄 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: Test that the `record_tokens` method of `_GeminiRateLimiter` returns early when no tokens are available.

Why Needed: This test prevents a potential regression where the limiter does not correctly handle cases with zero tokens.

Key Assertions:

- The length of `_token_usage` should be 0 after calling `record_tokens(0)`.
- The value of `_token_usage` should be an empty list.
- _token_usage is expected to be a list containing one element, which is an empty list.
- The limiter's internal state should not have been modified by the test.
- No new tokens are added to the limiter's internal queue during this test.
- The `record_tokens` method does not throw any exceptions when called with zero tokens.
- _GeminiRateLimiter's record_tokens method is expected to correctly handle cases with zero 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`	6 lines (ranges: 39-42, 66-67)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify that the `GeminiRateLimit` instance raises a `RateLimitExceeded` exception when exceeding the daily limit.

Why Needed: Prevents regression where the `GeminiRateLimiter` instance does not raise an error when exceeding the daily rate limit, potentially causing unexpected behavior or errors in downstream applications.

Key Assertions:

- The `wait_for_slot` method of the `GeminiRateLimit` instance raises a `RateLimitExceeded` exception with the specified message.
- The `requests_per_day` attribute of the `_GeminiRateLimitConfig` object is set to 1, which means the daily limit is 1 request per day.
- The `record_request` method of the `_GeminiRateLimiter` instance records a valid request before attempting to wait for a slot.
- The `wait_for_slot` method attempts to wait for a slot with an ID greater than 0, which is not possible when the daily limit has been exceeded.
- The `requests_per_day` attribute of the `_GeminiRateLimitConfig` object does not allow exceeding the daily limit (1 in this case).
- The `record_request` method records a valid request before attempting to wait for a slot, ensuring that the rate limit is respected even when exceeding 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/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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the TPM fallback wait is correctly implemented when filling up TPM.

Why Needed: This test prevents a regression where the TPM limiter does not handle cases where it needs to wait for a longer period of time to fill up the available tokens.

Key Assertions:

- The total number of tokens used by the request is greater than or equal to the rate limit.
- _tokens_used >= _rate_limit
- request_tokens > _limit
- token_usage is not empty

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 380-381, 384, 388-390)

PASSED tests/test_gemini_coverage_v2.py::test_gemini_provider_rpm_cooldown 559ms 🛡️ 6

AI ASSESSMENT

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

Why Needed: To prevent a potential issue where the RPM rate limit is not properly reset after a failed request.

Key Assertions:

- The 'models/gemini-pro' model should be present in the provider's cooldowns.
- The cooldown for 'models/gemini-pro' should exceed 1000 seconds (1 minute).
- The RPM rate limit should not be exceeded on subsequent calls to the same provider.
- The provider's cooldowns should contain a key matching 'models/gemini-pro'.
- The provider's cooldowns should have a value greater than 1000 seconds (1 minute).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

📄 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 model with an error message when rate limiting is triggered.

Why Needed: This test prevents regression in case of rate limiting issues, ensuring that models are annotated correctly even if they fail due to rate limits.

Key Assertions:

- The annotation should have a scenario attribute set to 'Recovered Scenario'.
- The error message should be None. If it's not None, the test fails.
- The mock_post call count should be equal to 2. If it's less than 2, the test fails.
- The provider._parse_response method should return a Mock object with a scenario attribute set to 'Recovered Scenario'.
- The mock_parse method should not have been called during the test.
- The time.sleep function should not be called during the test.
- The annotation should only contain one part, which is a text string. If it contains multiple parts, the test fails.
- The annotation content should match the expected content. The actual content may differ due to the mock data.
- The provider._annotate_internal method should call the _parse_response method and then return the annotated result. If this step is skipped or not called correctly, the test 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/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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that _annotate_success correctly annotates a success response from GeminiProvider

Why Needed: To prevent regression in case the actual _call_gemini returns an error.

Key Assertions:

- assert annotation.scenario == 'Success Scenario'
- assert not annotation.error
- assert mock_parse.return_value.scenario == '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: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the provider does not check for availability when GEMINI_API_TOKEN is set.

Key Assertions:

- The `_check_availability()` method of the `GeminiProvider` class should return False when `GEMINI_API_TOKEN` is set.
- The `_check_availability()` method of the `GeminiProvider` class should return True when `GEMINI_API_TOKEN` is not set.
- When GEMINI_API_TOKEN is set, the provider's availability check should be successful.
- When GEMINI_API_TOKEN is not set, the provider's availability check should fail.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the rate limiter allows more than one request in a single day, potentially causing unexpected behavior or errors.

Key Assertions:

- The next_available_in method returns None when there are no available slots in the rate limit.
- The limiter records a request before checking if it can be added to the rate limit.
- The limiter checks for available slots in the rate limit and prevents adding more requests than allowed per day.
- The limiter does not allow multiple requests in a single day, preventing potential concurrency issues.
- The limiter ensures that each request is checked individually before being added to the rate limit.
- The limiter records all requests made within the time frame of the daily limit.
- The limiter prevents adding more requests than allowed per day by checking for available slots in the rate limit.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/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: 380-381, 384, 388-390)

PASSED tests/test_gemini_provider.py::TestGeminiRateLimiter::test_rpm_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	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: 380-381, 384, 388-390)

📄 tests/test_hashing.py

13 tests

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

AI ASSESSMENT

Scenario: Test that different configuration providers produce different hashes.

Why Needed: This test prevents a potential bug where the same configuration is used with different providers, potentially leading to unexpected behavior or incorrect results.

Key Assertions:

- The function `compute_config_hash` should return a different hash for two different configuration objects.
- The hash of `config1` should not be equal to the hash of `config2`.
- The hash of `config1` should not be equal to the hash of `Config(provider='ollama')` (which is presumably an empty Config object).
- The hash of `config2` should not be equal to the hash of `Config(provider='none')` (which is presumably an empty 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`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`src/pytest_llm_report/util/hashing.py`	8 lines (ranges: 96-101, 103-104)

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

AI ASSESSMENT

Scenario: Verifies that the computed configuration hash is short and returns 16 characters.

Why Needed: Prevents a potential issue where the hash length exceeds 16 characters, potentially causing issues with storage or transmission.

Key Assertions:

- The length of the computed configuration hash should be exactly 16 characters.
- The computed configuration hash should not exceed 16 characters in length.
- The computed configuration hash should contain only ASCII characters (i.e., no non-ASCII characters).
- No leading zeros should appear in the computed configuration hash.
- No trailing zeros should appear in the computed configuration hash.
- All digits should be present in the computed configuration hash, without any leading zeros or trailing zeros.
- The computed configuration hash should not contain any non-numeric characters (e.g., punctuation, whitespace).
- The computed configuration hash should only contain numeric characters (0-9).

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`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 verifies that the hash of a file matches its content hash when written to disk.

Why Needed: This test prevents a potential bug where the hash of a file is not consistent with its content hash after being written to disk, which could lead to incorrect identification of files as modified even if they haven't changed.

Key Assertions:

- The file hash computed from the file's content should match the same hash computed from the entire file.
- The file path should be a valid temporary directory path.
- The file should not have been deleted or moved between test runs.
- The file's content should remain unchanged after being written to disk.
- The file's hash should not have changed even if its content has changed.
- The file path should still point to the same location on disk.
- The file should not have been modified since it was last written to disk.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`src/pytest_llm_report/util/hashing.py`	`6 lines (ranges: 32, 44-48)`

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

AI ASSESSMENT

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

Why Needed: This test prevents potential issues where incorrect hashing is returned due to incorrect file contents or encoding.

Key Assertions:

- The length of the computed hash should be 64 bytes.
- The computed hash should match the expected value (in this case, 'hello world' encoded in hexadecimal).

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`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 two different keys may generate the same HMAC signature, which could lead to data corruption or security vulnerabilities.

Key Assertions:

- The computed HMAC signature for the given input content and key 'key1' is different from the computed HMAC signature for the given input content and key 'key2'.
- The computed HMAC signature for the given input content 'content' and key 'key1' is different from the computed HMAC signature for the given input content 'content' and key 'key2'.
- The computed HMAC signature for the given input content 'content' and key 'key1' is not equal to the computed HMAC signature for the given input content 'content' and key 'key3'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 61)

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

AI ASSESSMENT

Scenario: Verifies that the `compute_hmac` function produces an HMAC of the correct length when given a secret key.

Why Needed: Prevents a potential issue where the HMAC is not produced with the expected 64-byte length, potentially leading to incorrect data integrity checks.

Key Assertions:

- The output of `compute_hmac(b'content', b'secret-key')` should be a bytes object with a length of 64 bytes.
- The output of `compute_hmac(b'content', b'secret-key')` should not have any trailing zeros (i.e., it should be a hexadecimal string without leading zeros).
- The output of `compute_hmac(b'content', b'secret-key')` should only contain the characters 'a' through 'z' and 'A' through 'Z'.
- The output of `compute_hmac(b'content', b'secret-key')` should not be empty.
- The output of `compute_hmac(b'content', b'secret-key')` should be a hexadecimal string without leading zeros or trailing spaces.
- The output of `compute_hmac(b'content', b'secret-key')` should only contain the characters 'a' through 'z' and 'A' through 'Z'.
- The output of `compute_hmac(b'content', b'secret-key')` should not be empty or have any leading zeros.
- The output of `compute_hmac(b'content', b'secret-key')` should only contain the characters 'a' through 'z' and 'A' through 'Z'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`src/pytest_llm_report/util/hashing.py`	`1 lines (ranges: 61)`

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

AI ASSESSMENT

Scenario: The test verifies that the SHA-256 hash of two identical input bytes is also identical.

Why Needed: This test prevents a potential bug where different inputs to `compute_sha256` could produce different hashes due to timing or other factors.

Key Assertions:

- Two identical input bytes should produce the same output hash.
- The output hash of two identical input bytes should be equal to the expected output.
- The hash value should not change when the same input is used multiple times with the same algorithm instance.
- Different inputs to `compute_sha256` should produce different hashes due to timing or other factors.
- The hash value should remain unchanged even if the same input is used in a different order.
- The hash value should be consistent across different Python versions and platforms.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	1 lines (ranges: 32)

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

AI ASSESSMENT

Scenario: The hash function computes the SHA-256 of a given string and verifies that it is 64 hexadecimal characters long.

Why Needed: This test prevents a potential bug where the hash length is not correctly calculated or reported, potentially leading to incorrect identification of valid inputs.

Key Assertions:

- The computed hash should be exactly 64 bytes (128 bits) in length.
- The computed hash should consist only of hexadecimal digits (0-9, A-F, a-f).
- The computed hash should not contain any whitespace characters or special characters.
- The computed hash should start with the expected prefix '0x'.
- The computed hash should be a valid SHA-256 digest.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`src/pytest_llm_report/util/hashing.py`	`1 lines (ranges: 32)`

PASSED `tests/test_hashing.py::TestGetDependencySnapshot::test_includes_pytest` 93ms 🛡️ 3

AI ASSESSMENT

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

Why Needed: This test prevents a regression where the dependency snapshot is not correctly populated with pytest-related dependencies.

Key Assertions:

- The 'pytest' package should be included in the dependency snapshot.
- The 'pytest' package should be present in the snapshot's list of dependencies.
- The snapshot's list of dependencies should include the 'pytest' package.
- The snapshot's 'pytest' field should contain a boolean value indicating its presence.
- The 'pytest' package should not be included if it is not installed or available.
- The snapshot's dependencies should not be empty when pytest 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/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	8 lines (ranges: 113-114, 116-121)

PASSED `tests/test_hashing.py::TestGetDependencySnapshot::test_returns_dict` 88ms 🛡️ 3

AI ASSESSMENT

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

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

Key Assertions:

- snapshot is not None
- snapshot is a dictionary
- snapshot has a length greater than 0
- all values in snapshot are dictionaries
- all values in snapshot have keys that start with 'package_'
- the function does not raise an exception when the dependency snapshot is empty

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`src/pytest_llm_report/util/hashing.py`	8 lines (ranges: 113-114, 116-121)

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

AI ASSESSMENT

Scenario: The test verifies that the `load_hmac_key` function correctly loads a key from a file.

Why Needed: This test prevents a bug where the HMAC key is not loaded correctly if the file does not exist or is corrupted.

Key Assertions:

- The `load_hmac_key` function should be able to load the HMAC key from the specified file.
- The `load_hmac_key` function should raise an exception when the file does not exist or is corrupted.
- The `load_hmac_key` function should return the correct HMAC key for the loaded configuration.
- The `load_hmac_key` function should handle errors properly and provide informative error messages.
- The `load_hmac_key` function should not modify the original file if it already contains a valid key.
- The `load_hmac_key` function should support loading keys from different types of files (e.g., JSON, YAML).
- The `load_hmac_key` function should be able to handle large files without running out of memory.
- The `load_hmac_key` function should provide a way to verify the integrity of the loaded key.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	5 lines (ranges: 73, 76-77, 80-81)

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

AI ASSESSMENT

Scenario: Test 'test_missing_key_file' verifies that the function returns None when a missing key file is provided.

Why Needed: This test prevents a potential bug where the function would return an incorrect result if a key file with the expected name does not exist.

Key Assertions:

- The function `load_hmac_key(config)` should be called with a valid path to a key file.
- The function `load_hmac_key(config)` should raise a `KeyFileNotFoundError` when the provided key file does not exist.
- The function `load_hmac_key(config)` should return None for the given key file path.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/hashing.py	4 lines (ranges: 73, 76-78)

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

AI ASSESSMENT

Scenario: Test 'test_no_key_file' verifies that the function returns None when no key file is specified.

Why Needed: This test prevents a potential bug where the function does not return an error message or None when no key file is provided.

Key Assertions:

- The function should be able to successfully load the HMAC key from the configuration without any errors.
- The function should return None as expected when no key file is specified in the configuration.
- The function should not throw an exception or raise an error when no key file is provided.
- The function should handle invalid input correctly (e.g., empty string, null value) and still return None.
- The function should follow best practices for handling missing or invalid data (e.g., logging an error message)
- The function should not silently fail without providing any useful feedback to the user
- The function should be able to handle different types of key files (e.g., PEM, JSON) correctly

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`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 are not set correctly.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify that the default capture failed output is set to False.

Why Needed: This test prevents a potential bug where the default configuration does not allow for capturing of failed outputs.

Key Assertions:

- config.capture_failed_output should be set to False
- config.capture_failed_output is False
- capture_failed_output is opt-in, i.e., it's off by default

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	3 lines (ranges: 107, 147, 233)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The LLM is not enabled by default in the test configuration.

Why Needed: This test prevents a regression where the default LLM configuration is disabled.

Key Assertions:

- config.is_llm_enabled() should return False
- config.get_llm_enabled_value() should return False
- get_default_config().llm_enabled() should be False
- get_default_config().llm_enabled_value() should be False
- is_llm_enabled() method should throw an exception when called on a 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`	4 lines (ranges: 107, 147, 224, 233)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify that the `get_default_config()` function returns a configuration object with an `omit_tests_from_coverage` attribute set to `True`.

Why Needed: This test prevents a bug where the default configuration does not include tests by omitting them from coverage analysis.

Key Assertions:

- The `config` variable is assigned a value that matches the expected result of calling `get_default_config()`.
- The `omit_tests_from_coverage` attribute of the `config` object is set to `True` as expected.
- The `config` object has the correct attributes, including `omit_tests_from_coverage`.
- The `get_default_config()` function returns a configuration object with the required attributes.
- The `get_default_config()` function does not raise an exception when called without arguments.
- The `get_default_config()` function can be called multiple times without affecting the result.
- The `config` variable is reassigned to a new value after calling `get_default_config()`.
- The `config` object has the correct attributes, including `omit_tests_from_coverage`, even if it is reassigned.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: Prevents a potential bug where the provider is not set to 'none' when it is required for privacy.

Key Assertions:

- The `config.provider` attribute of the test configuration should be equal to 'none'.
- The `provider` key in the test configuration dictionary should contain the value 'none'.
- The `get_default_config()` function returns a test configuration with provider set to 'none'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `llm_context_exclude_globs` configuration parameter is set to include 'secret' files.
- The `llm_context_exclude_globs` configuration parameter is set to include '.env' files.
- Any secret file names are present in the excluded list.
- No sensitive information like secret files or .env files are included in the LLM context.
- The test ensures that only non-secret files and directories are processed by the LLM.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 233)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the deterministic output of the integration gate is reported correctly.

Why Needed: This test prevents a regression where the deterministic output may not be reported correctly due to sorting issues.

Key Assertions:

- The node IDs in the report are sorted in ascending order.
- The node IDs in the report are sorted in descending order.
- The node IDs in the report contain all unique values.
- The node IDs in the report do not contain any duplicates.
- The node IDs in the report are consistent across different runs.
- The node IDs in the report are sorted correctly even when there are duplicate node IDs.
- The node IDs in the report are sorted correctly even when there are no duplicate node IDs.

COVERAGE

src/pytest_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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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: Prevents regression in case of an empty test suite, ensuring the report is always accurate.

Key Assertions:

- The total count of tests is zero.
- There are no failed or skipped tests.
- All test results are included in the report.
- The report does not contain any errors.
- The report summary includes a 'total' key with a value of zero.
- Each test result is represented as an object with 'test_name', 'status', and 'message' keys.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	118 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-471, 495, 497, 499-501, 503, 506)

PASSED `tests/test_integration_gate.py::TestFullPipeline::test_html_report_generation` 34ms 🛡️ 6

AI ASSESSMENT

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

Why Needed: This test prevents regression due to a change in the configuration of the report writer, which may have caused the report not to be generated correctly.

Key Assertions:

- The file "report.html" exists at the specified path.
- The string '
- The string 'test_pass' is present in the content of the file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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` 57ms 🛡️ 7

AI ASSESSMENT

Scenario: The test verifies that a full pipeline generates a valid JSON report.

Why Needed: This test prevents regression where the JSON report is not generated correctly due to an issue with the report writer configuration.

Key Assertions:

- The path of the generated JSON report should exist at `tmp_path / 'report.json'`.
- The schema version of the JSON report should be `SCHEMA_VERSION`.
- The summary statistics of the JSON report should match the expected values: `total=3`, `passed=1`, `failed=1`, and `skipped=1`.

COVERAGE

src/pytest_llm_report/_git_info.py	2 lines (ranges: 2-3)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)

`src/pytest_llm_report/report_writer.py`	133 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226-227, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-322, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 376, 378-379, 382, 385, 388, 391-395, 470-471, 495, 497, 499-501, 503, 506)

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

AI ASSESSMENT

Scenario: Verify that the ReportRoot object has required fields.

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

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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify that `RunMeta` has 'aggregation_fields' in its metadata.

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

Key Assertions:

- The 'is_aggregated' field is present in the data.
- The 'run_count' field is present in the data.
- The 'aggregation_policy' field is included when 'is_aggregated' is True.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`29 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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

AI ASSESSMENT

Scenario: Test `RunMeta` has run status fields.

Why Needed: Prevents regression where `RunMeta` does not have a 'status' field.

Key Assertions:

- The presence of 'exit_code', 'interrupted', and 'collect_only' keys in the data.
- The presence of 'collected_count' and 'selected_count' keys 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: 380-381, 384, 388-390)

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 present in the string.
- The schema version should contain at least one dot (.) character.
- The schema version should be a valid semver-like 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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test 'test_case_has_required_fields' verifies that the TestCaseResult object has required fields.

Why Needed: This test prevents a potential bug where the TestCaseResult object is missing required fields, potentially causing unexpected behavior or errors.

Key Assertions:

- The 'nodeid' field should be present in the TestCaseResult object.
- The 'outcome' field should be present in the TestCaseResult object.
- The 'duration' field should be present in the TestCaseResult object.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/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: 380-381, 384, 388-390)

📄 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 a `GeminiProvider` instance when the 'provider' parameter is set to 'gemini'.

Why Needed: This test prevents a potential bug where the correct provider is not returned for the 'gemini' configuration.

Key Assertions:

- The function `get_provider(config)` should return an instance of `GeminiProvider`.
- The class name of the returned `GeminiProvider` instance should be `GeminiProvider`.
- The method name of the returned `GeminiProvider` instance should be `__class__ == 'GeminiProvider'`.
- The attribute name of the returned `GeminiProvider` instance should be `__class__.__name__`.
- The value of the `provider` parameter in the test configuration should be 'gemini'.
- The provider name in the test configuration should be 'gemini'.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	10 lines (ranges: 52-53, 245, 247, 249, 252, 257, 262-263, 265)
`src/pytest_llm_report/llm/gemini.py`	7 lines (ranges: 134, 136-139, 141-142)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the `get_provider` function does not return an instance of `LiteLLMProvider` when a non-supported provider is used.

Key Assertions:

- The `__class__.__name__` attribute of the returned `provider` object should be equal to `'LiteLLMProvider'`.
- The `provider` object should have an attribute named `model` with value `'gpt-3.5-turbo'`.
- The `provider` object should have a method named `__init__` that accepts the `provider` parameter.
- The `provider` object should be an instance of `LiteLLMProvider` when it is created using the `get_provider` function.
- The `provider` object should not be None when it is created using the `get_provider` function.
- The `provider` object should have a valid provider name set to `'litellm'`.
- The `provider` object should have a valid model name set to `'gpt-3.5-turbo'`.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	9 lines (ranges: 52-53, 245, 247, 249, 252, 257-258, 260)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

Key Assertions:

- config.provider should be set to 'none'
- provider should be an instance of NoopProvider
- assert isinstance(provider, NoopProvider) should be true

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	6 lines (ranges: 52-53, 245, 247, 249-250)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the OllamaProvider is returned when the 'ollama' provider is specified.

Why Needed: This test prevents a potential bug where the correct provider (Ollama) is not selected.

Key Assertions:

- assert provider.__class__.__name__ == 'OllamaProvider',
- assert isinstance(provider, OllamaProvider),
- assert provider.model == 'llama3.2',
- assert provider.name == 'ollama',
- assert provider.id == 1,
- assert provider.config is not None and config.provider == 'ollama',
- assert isinstance(config, Config),
- assert config.model == 'llama3.2' and config.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: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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

AI ASSESSMENT

Scenario: Test that an unknown provider raises a ValueError when trying to get a provider.

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

Key Assertions:

- The function `get_provider` should raise a `ValueError` with the message 'unknown' when called with an unknown provider.
- The error message should contain the string 'unknown'.
- The error message should be case-insensitive (e.g., 'Unknown Provider' instead of 'UNknown Provider').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 245, 247, 249, 252, 257, 262, 267)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that NoopProvider implements LlmProvider interface.

Why Needed: Prevents regression in case of missing required methods.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the annotate method does not return an annotation if no scenario is specified.

Key Assertions:

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

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The NoopProvider instance should always be available.

Why Needed: To prevent a potential bug where the provider might not be initialized before being used.

Key Assertions:

- provider.is_available() should return True
- config is not None
- provider is an instance of NoopProvider
- provider is not None after calling is_available()
- NoopProvider instance creation succeeds

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/base.py`	`5 lines (ranges: 52-53, 107, 110-111)`
`src/pytest_llm_report/llm/noop.py`	`2 lines (ranges: 32, 58)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

📄 tests/test_llm_annotator.py

6 tests

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

AI ASSESSMENT

Scenario: Test that annotation summary is printed when annotations run.

Why Needed: This test prevents regression where annotation summaries are not printed for annotated tests.

Key Assertions:

- The function `get_provider` from `pytest_llm_report.llm.annotator` returns a `FakeProvider` instance.
- The `FakeProvider` instance is set as the provider for annotations using `monkeypatch.setattr`.
- An annotation summary is printed when an annotated test is run, which includes 1 test(s) via litellm.
- The captured output from `capsys.readouterr()` contains 'Annotated 1 test(s) via litellm'.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	65 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

Scenario: Test LLM annotator progress reporting for multiple tests.

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

Key Assertions:

- The test verifies that the LLM annotation progress is reported correctly for all tests.
- The test checks if the correct messages are appended to the `messages` list.
- The test asserts that the first message in the `messages` list contains the expected string.
- The test checks if the second message in the `messages` list contains the expected string.
- The test verifies that the progress is reported for all tests.
- The test checks if the correct provider is used to get the LLM annotator instance.
- The test asserts that the `append` method of the `messages` list returns a non-empty list.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	69 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-68, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192-195, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 that LLM annotations do not skip opt-out tests and respect the maximum number of tests.

Key Assertions:

- The 'tests/test_a.py::test_a' node should be called with an LLM annotation.
- The first test case should have an LLM annotation.
- The second test case should not have an LLM annotation (opt-out is True).
- The third test case should not have an LLM annotation (opt-out is True).
- The 'tests/test_a.py::test_a' node should be called with the correct number of tests (1).
- The maximum number of tests should not exceed 1.
- The LLM annotator should not call any other provider for the test case.
- The LLM annotation should only be created once for the test case.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	65 lines (ranges: 45, 48-49, 56-57, 59, 61-62, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-171, 173-174, 176, 178, 180, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

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

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

Key Assertions:

- The provider's calls to `llm.annotator.get_provider` should match the expected list of node IDs.
- The `time.sleep` function should be called twice with times [0.0, 2.0].
- The number of requests made by the LLM annotator within a minute should not exceed the allowed limit of 30 requests per minute.

COVERAGE

src/pytest_llm_report/cache.py	20 lines (ranges: 39-41, 53, 55-56, 86, 90, 92, 94, 97-101, 103, 118-119, 121, 153)
src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	68 lines (ranges: 45, 48-49, 56-57, 59, 61, 64, 66-67, 71-72, 74-78, 87-92, 97-98, 100, 102, 104, 115-122, 129-135, 137, 139, 165-168, 170-173, 176, 178, 180-183, 185-190, 192, 198, 203)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	12 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-84)

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

AI ASSESSMENT

Scenario: Tests should be skipped when unavailable providers are used.

Why Needed: This test prevents regression and ensures that annotation is skipped when an unavailable provider is encountered.

Key Assertions:

- The `is_available` method of the `UnavailableProvider` class returns False.
- The `get_provider` function from `pytest_llm_report.llm.annotator` sets the `provider` attribute to the `UnavailableProvider` instance.
- The `annotate_tests` function checks if the provider is available before proceeding with annotation.
- If an unavailable provider is encountered, the test should skip annotation and report a message indicating that it's not available.
- The captured output from the test should contain the string 'is not available' to verify this assertion.
- The `tmp_path` parameter passed to the function ensures that the test uses a temporary directory for testing purposes.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/annotator.py	7 lines (ranges: 45, 48-52, 54)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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_required_fields` function checks if the schema requires 'scenario' and 'why_needed' fields.

Why Needed: This test prevents a regression where the LLM contract's schema does not require these two critical fields.

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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the AnnotationSchema.from_dict method correctly parses a dictionary into an AnnotationSchema instance.

Why Needed: This test prevents authentication bypass by ensuring that the annotation is properly configured with the correct scenario, why needed, and key assertions.

Key Assertions:

- checks password
- checks username

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the AnnotationSchema handles an empty input by asserting its default scenario and expected reason.

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

Key Assertions:

- schema.scenario == ""
- schema.why_needed == ""
- assert schema.scenario == ""
- assert schema.why_needed == ""
- assert isinstance(schema.scenario, str)
- assert isinstance(schema.why_needed, str)
- assert isinstance(schema.scenario, str) and isinstance(schema.why_needed, str)

COVERAGE

src/pytest_llm_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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: test_schema_handles_partial verifies that the AnnotationSchema from_dict method correctly handles a partial input.

Why Needed: This test prevents regression in handling invalid or incomplete input data, ensuring the correct behavior of the AnnotationSchema.

Key Assertions:

- The schema's scenario attribute is set to "Partial only".
- The schema's why_needed attribute is empty.
- The schema correctly validates and returns an empty string for a partial input.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/schemas.py	5 lines (ranges: 77-81)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential bug where the annotation schema is not properly defined with required fields.

Key Assertions:

- assert 'scenario' in ANNOTATION_JSON_SCHEMA['properties']
- assert 'why_needed' in ANNOTATION_JSON_SCHEMA['properties']
- assert 'key_assertions' in ANNOTATION_JSON_SCHEMA['properties']
- assert isinstance(ANNOTATION_JSON_SCHEMA, dict)
- assert ANNOTATION_JSON_SCHEMA is not None
- assert ANNOTATION_JSON_SCHEMA != {}
- assert all(key in ANNOTATION_JSON_SCHEMA for key in ['scenario', 'why_needed', 'key_assertions'])

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_llm_contract.py::TestAnnotationSchema::test_schema_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/llm/schemas.py	7 lines (ranges: 90-92, 94-96, 98)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 the NoopProvider instance is not returned correctly for the 'provider' parameter set to 'none'.

Key Assertions:

- The function `get_provider(config)` should return an instance of `NoopProvider` when the `config` object has a `provider` attribute set to `'none'`.
- The `provider` attribute in the `config` object should be set to `'none'` for the returned NoopProvider instance.
- The `get_provider(config)` function should not raise an exception or return None when the `config` object has a valid provider.
- The `NoopProvider` class should have a constructor that accepts a `provider` parameter and sets it to `'none'` if the `provider` attribute is set to `'none'`.
- The `get_provider(config)` function should return an instance of `NoopProvider` with the correct provider name.
- The `get_provider(config)` function should not throw any errors when called with a valid configuration object.
- The `Config` class should have a method that returns a `provider` attribute set to `'none'` for a valid configuration object.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	6 lines (ranges: 52-53, 245, 247, 249-250)
`src/pytest_llm_report/llm/noop.py`	1 lines (ranges: 32)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that a NoopProvider instance can be instantiated and returned as an instance of LlmProvider.

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

Key Assertions:

- A NoopProvider instance should be an instance of Config.
- A NoopProvider instance should not implement LlmProvider.
- The returned provider should have the correct type.
- NoopProvider instances do not raise any errors when instantiated.
- NoopProvider instances do not attempt to use the underlying config.
- The NoopProvider class does not inherit from LlmProvider.
- The NoopProvider class has a different constructor than LlmProvider.
- A NoopProvider instance is created with an empty 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/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/noop.py	1 lines (ranges: 32)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The NoopProvider should return an empty annotation when no node is found in the config.

Why Needed: This test prevents a regression where the NoopProvider returns a non-empty annotation when no node is found in the config.

Key Assertions:

- assert result.scenario == ""
- assert result.why_needed == ""
- assert result.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: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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 expected attributes.

Why Needed: This test prevents a potential regression where the annotate method does not return the expected LlmAnnotation-like object, potentially leading to incorrect results or errors in downstream tests.

Key Assertions:

- The result has the correct 'scenario' attribute.
- The result has the correct 'why_needed' attribute.
- The result has the correct '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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the `provider` handles a `None` context gracefully.

Why Needed: This test prevents potential bugs or regression in the contract by ensuring it can handle cases where `None` is passed as an argument to `annotate`.

Key Assertions:

- The `provider.annotate` method should not return `None` when a `None` context is provided.
- A `None` value should be returned instead of `None` for the 'code' key.
- No exception should be raised when passing a `None` context to `annotate`.
- The test result should not have an invalid or unexpected value.
- The `provider.annotate` method should return a valid object with the expected keys and values.
- A meaningful error message should be returned if the `None` context is passed to `annotate`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/noop.py	2 lines (ranges: 32, 50)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: Prevents regression in LLM contract when a new provider is added.

Key Assertions:

- The provider has an attribute named 'annotate'.
- The provider's 'annotate' function is callable.
- The provider's 'annotate' method does not raise any exceptions.
- The provider's 'annotate' method can be called without arguments.
- The provider's 'annotate' method can be called with a string argument.
- The provider's 'annotate' method can be called with an integer argument.
- The provider's 'annotate' method returns a value (e.g., a string or number).
- The provider's 'annotate' method does not raise any exceptions when called with invalid arguments.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	15 lines (ranges: 52-53, 245, 247, 249-250, 252-253, 255, 257-258, 260, 262-263, 265)
`src/pytest_llm_report/llm/gemini.py`	7 lines (ranges: 134, 136-139, 141-142)
`src/pytest_llm_report/llm/noop.py`	1 lines (ranges: 32)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

📄 tests/test_llm_providers.py

31 tests

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

AI ASSESSMENT

Scenario: The `test_annotate_handles_context_too_large` test verifies that the `annotate` method of the `GeminiProvider` class handles large contexts without raising an exception.

Why Needed: This test prevents a potential bug where the `annotate` method throws an exception when given extremely large contexts, causing the test to fail or hang.

Key Assertions:

- The context is annotated correctly and does not raise an exception.
- The annotation size is within the expected limit (e.g., 1024 bytes).
- The `annotate` method returns a valid object with the correct attributes.
- The test can be run without encountering errors or hang-ups due to excessive context sizes.

COVERAGE

src/pytest_llm_report/collector.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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The LiteLLM provider should report a missing dependency when trying to annotate a case that depends on it.

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

Key Assertions:

- The annotation should contain an error message indicating that 'litellm' is missing and how to install it.
- The error message should be specific to the case being annotated.
- The error message should include the exact name of the dependency ('litellm')
- The error message should not contain any misleading information (e.g., 'install 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/llm/base.py`	`5 lines (ranges: 52-53, 72, 75, 80)`
`src/pytest_llm_report/llm/gemini.py`	`12 lines (ranges: 134, 136-139, 141-142, 160-164)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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

AI ASSESSMENT

Scenario: Test that the GeminiProvider annotates missing tokens correctly.

Why Needed: This test prevents a potential bug where an annotator fails to recognize a missing token in the configuration.

Key Assertions:

- The annotation error is set to 'GEMINI_API_TOKEN is not set'.
- The provided function `test_case` is executed with the correct result.
- The `CaseResult` nodeid is correctly set to '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/gemini.py	12 lines (ranges: 134, 136-139, 141-142, 160-161, 167-169)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Verify that the `annotate` method records tokens on the limiter and rate limits correctly.

Why Needed: Prevents regressions due to incorrect token usage or rate limit enforcement.

Key Assertions:

- The 'json' key in the captured response is set to the provided JSON payload.
- The totalTokenCount in the response metadata matches the expected value of 123.
- The 'candidates' list contains a single dictionary with a 'text' key containing the response data.
- The 'usageMetadata' dictionary has a 'totalTokenCount' key with the expected value of 123.
- Rate limits are enforced correctly, and the number of tokens used does not exceed the allowed limit of 1000 per minute.
- Tokens are recorded on the limiter for the test function `test_login`.
- The rate limits logic is executed without raising an exception or returning an error.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the `annotate` method retries when rate limiting occurs.

Why Needed: This test prevents a potential regression where the `annotate` method does not retry when rate limiting is applied.

Key Assertions:

- The `annotate` method should retry after rate limiting is encountered.
- The retry count should be incremented for each failed annotation attempt.
- The maximum retry count should be capped at a reasonable value (e.g., 5-10 attempts).
- The test should fail with an error message indicating that rate limiting occurred, rather than silently failing.
- The `annotate` method should increment the retry count after each successful annotation attempt.
- The retry count should not exceed the maximum allowed value, even if all annotations succeed.
- The test should verify that the `annotate` method correctly handles rate limiting scenarios (e.g., 1-2 attempts per second).
- The test should pass with a reasonable delay between retries to simulate realistic usage scenarios.

COVERAGE

src/pytest_llm_report/collector.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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The `annotate_rotates_models_on_daily_limit` method should rotate models on the daily limit for a LLM provider.

Why Needed: This test prevents regression when the model rotation threshold is changed or updated.

Key Assertions:

- Rotates models on the daily limit by updating the underlying data structure.
- Updates the `rotate_models` method to reflect the new limit.
- Checks if the number of rotated models has reached the daily limit.
- Verifies that the `rotate_models` method is called with the correct threshold value.
- Ensures that the `rotate_models` method updates the model data correctly.
- Tests for any potential errors or exceptions raised during rotation.
- Verifies that the number of rotated models does not exceed the daily limit after rotation.
- Checks if the `rotate_models` method updates the underlying data structure 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	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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the annotate method skips daily limits when provided by a Gemini provider.

Why Needed: This test prevents a potential regression where the annotate method fails to skip daily limits due to a misconfigured provider.

Key Assertions:

- The annotate method should return an error or raise an exception when the daily limit is exceeded.
- The annotated text should not contain any information about the user's location.
- The annotated text should be empty if the daily limit has been skipped.
- The annotated text should only contain the provided annotation.
- The annotated text should not include any additional metadata or context.
- The annotated text should not have any links or attachments.
- The annotated text should not include any user-generated content.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/gemini.py	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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: Prevents regressions by ensuring correct annotation of responses with expected data.

Key Assertions:

- status ok
- redirect
- model gpt-4o
- system role
- tests/test_auth.py::test_login

COVERAGE

src/pytest_llm_report/collector.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: 107, 147)

src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The model's performance metrics (e.g. accuracy, F1 score) should improve over time.
- The model's inference times should decrease as it recovers from exhaustion.
- The model's memory usage should decrease as it recovers from exhaustion.
- The model's latency should decrease as it recovers from exhaustion.
- The model's training process should complete within 24 hours if it was previously failing due to exhaustion.
- The model's inference times should be within a reasonable range (e.g. < 1 second) after 24 hours of recovery.

COVERAGE

src/pytest_llm_report/collector.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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the `fetch_available_models` method returns an error when no models are available.

Why Needed: This test prevents a regression where the `fetch_available_models` method does not return an error for cases where there are no available models.

Key Assertions:

- assertRaises(SystemExit), SystemExit
- assertRaises(NotImplementedError), NotImplementedError
- assertRaises(ValueError), 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	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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The `refresh_interval` attribute of the provider is set to the expected value.
- The `model_list` attribute is updated with the latest models within the specified time frame.
- The `refreshed_models` attribute contains all the models that were added or removed during the interval.
- The `time elapsed` attribute is greater than or equal to the `refresh_interval` attribute.
- All models in the `model_list` are present in the `refreshed_models` set.
- No models are missing from the `refreshed_models` set.
- The `refreshed_models` set contains all unique models that were added or removed during the interval.

COVERAGE

src/pytest_llm_report/collector.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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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: To prevent a regression where LiteLLM providers do not surface completion errors in annotations.

Key Assertions:

- The annotation should contain the string 'boom' as an error.
- The annotation should indicate that the function `test_case` raises a RuntimeError.
- The annotation should include the actual error message 'boom'.
- The annotation should not ignore the completion error and report it as an error.
- The annotation should not silently raise the exception, but instead propagate it to the caller.

COVERAGE

src/pytest_llm_report/collector.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	18 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 80, 82)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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.

Key Assertions:

- response_data must be a dictionary
- response_data must contain 'key_assertions' as a list
- response_data must not contain any other keys besides 'scenario', 'why_needed', and 'key_assertions'
- response_data must not have any duplicate key_assertions
- response_data must contain at least one valid key_assertion
- response_data must be of the correct type (dict)
- response_data must not have any invalid values for '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	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	21 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 66-67, 69, 71, 76)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The annotation contains the correct error message indicating that 'litellm' is missing and how to install it.
- The annotation includes the correct path to the installation command for 'litellm'.
- The annotation does not report a dependency that does not exist (i.e., no error is raised).

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	5 lines (ranges: 52-53, 72, 75, 80)
`src/pytest_llm_report/llm/litellm_provider.py`	5 lines (ranges: 37-41)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that the LiteLLM provider annotates a successful response with the correct information.

Why Needed: Prevents regressions by verifying that the provider correctly parses and annotates valid responses.

Key Assertions:

- status ok
- redirect
- model gpt-4o
- role system
- tests/test_auth.py::test_login

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	26 lines (ranges: 52-53, 72, 75, 80, 165, 167, 175, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
`src/pytest_llm_report/llm/litellm_provider.py`	20 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 66-67, 69, 78)
`src/pytest_llm_report/llm/schemas.py`	7 lines (ranges: 38, 42-43, 50-53)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	5 lines (ranges: 52-53, 107, 110-111)
src/pytest_llm_report/llm/litellm_provider.py	3 lines (ranges: 99-100, 102)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a potential regression where the annotate function fails to work as expected when encountering a context length error.

Key Assertions:

- It should not raise an exception when the context length is too long.
- It should return a default value (e.g., 'default') instead of raising an exception or returning None.
- It should handle the case where the context length is exactly equal to the maximum allowed length.
- The function should log an error message with the context length and reason for the error.
- The function should not raise a TypeError when the input is not a valid annotation type.
- The function should return a default value (e.g., 'default') instead of raising an exception or returning None when the input is invalid.
- The function should handle the case where the input is an empty list or tuple.
- The function should log a warning message with the reason for the error, but not raise an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The test verifies that the annotate method returns an error message when a call to Ollama raises a call error.

Why Needed: This test prevents regression in case the Ollama provider fails to handle call errors correctly.

Key Assertions:

- The annotation should return 'Failed after 2 retries. Last error: boom' as the error message when a call to Ollama raises a call error.
- The annotation should set the `error` attribute of the annotation object to 'Failed after 2 retries. Last error: boom'.
- The annotation should not raise an exception when a call to Ollama raises a call error, 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`	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: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The Ollama provider should report an error when missing the httpx dependency.

Why Needed: This test prevents a bug where the provider does not report an error for missing dependencies, potentially leading to unexpected behavior or silent failures.

Key Assertions:

- assert annotation.error == 'httpx not installed. Install with: pip install httpx'
- assert mock_import_error('httpx') was called exactly once
- assert test_case().outcome == '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/llm/base.py	5 lines (ranges: 52-53, 72, 75, 80)
src/pytest_llm_report/llm/ollama.py	5 lines (ranges: 40-44)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: Prevents potential authentication bugs in the annotating process.

Key Assertions:

- Check if the status of the response matches 'Tests user login'
- Validate that a token is present in the response
- Verify that the response contains the expected JSON structure

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test Ollama provider makes correct API call when calling ollama with a valid model and prompt.

Why Needed: This test prevents regression where the Ollama provider fails to make an API call due to incorrect model or prompt provided.

Key Assertions:

- The API call is made successfully with a status code of 200.
- The response from the API is 'test response'.
- The model used in the API call is correctly set to 'llama3.2:1b'.
- The prompt provided for generation is correctly set to 'test prompt'.
- The system prompt provided for generation is correctly set to 'system prompt'.
- The stream parameter is not set to True.
- The timeout is set to 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/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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that Ollama provider uses default model when not specified.

Why Needed: This test prevents regression where the default model is used instead of a custom one.

Key Assertions:

- The captured response from Ollama should contain 'model': 'llama3.2' key.
- The captured response from Ollama should not have any custom JSON data.
- The captured response from Ollama should be in the format of a valid HTTP response object with 'response' key.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	2 lines (ranges: 52-53)
src/pytest_llm_report/llm/ollama.py	16 lines (ranges: 119, 121-128, 132-135, 137, 139-140)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 Ollama provider incorrectly reports availability even when the server is down.

Key Assertions:

- The `OllamaProvider` instance does not call `_check_availability()` with a truthy value.
- The `OllamaProvider` instance raises an exception (`ConnectionError`) instead of returning False.
- The `Config` instance passes the correct provider to the `OllamaProvider` constructor.
- The `provider` attribute is set to the fake `SimpleNamespace` instance with a `get()` function that raises a `ConnectionError`.
- The `_check_availability()` method is called on the fake `SimpleNamespace` instance and returns False.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	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: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents a bug where the provider incorrectly assumes a successful response (status code 200) when it's not available.

Key Assertions:

- The method _check_availability() of the provider is called.
- The status code returned by the provider is not 200.
- The provider returns False for non-200 status codes.
- The provider does not raise an exception when a non-200 status code is encountered.
- The provider's response is not a dictionary or a list.
- The provider's response contains only strings.
- The provider's response contains only integers.
- The provider's response contains other types of data (e.g., 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/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: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test checks availability of Ollama provider via /api/tags endpoint.

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

Key Assertions:

- The '/api/tags' URL is present in the provided URL.
- The response status code is 200 (OK).
- The response contains the expected 'tags' key in its JSON payload.
- The provider's availability check returns True for the specified configuration and host.
- The provider does not respond to requests for tags with a 404 status code.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/llm/base.py`	`2 lines (ranges: 52-53)`
`src/pytest_llm_report/llm/ollama.py`	`5 lines (ranges: 92-93, 95-97)`
`src/pytest_llm_report/options.py`	`2 lines (ranges: 107, 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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

AI ASSESSMENT

Scenario: The Ollama provider should always return `is_local=True`.

Why Needed: To prevent regression and ensure the correctness of the `OllamaProvider` class.

Key Assertions:

- provider.is_local() == True
- provider.config.provider == 'ollama'
- provider._local is False
- config._provider == 'ollama'
- provider._local is not None
- provider._local != False
- OllamaProvider.__init__(config) was called with correct arguments

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/base.py`	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: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

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

Key Assertions:

- The method `_parse_response()` of the `OllamaProvider` class should throw an exception when receiving an invalid JSON string.
- The error message 'Failed to parse LLM response as JSON' should be thrown by the `_parse_response()` method.
- When the input is a string containing invalid JSON, the expected error message should be raised.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	7 lines (ranges: 52-53, 186-187, 190-192)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-52, 55)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_llm_providers.py::TestOllamaProvider::test_parse_response_invalid_key_assertions` 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	16 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207-209)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

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

Why Needed: This test prevents potential bugs where the provider fails to extract JSON from markdown code fences, potentially leading to incorrect or incomplete annotations.

Key Assertions:

- The extracted JSON is not empty.
- The JSON string contains only valid JSON syntax.
- The JSON object has a 'text' key with the correct value.
- The JSON object has an 'annotations' key with the expected structure and values.
- The provider correctly handles nested objects and arrays in the JSON response.
- The extracted JSON is not null or undefined.
- The JSON string does not contain any invalid characters or syntax errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	6 lines (ranges: 38, 42-44, 46-47)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: The provided test verifies that the Ollama provider can extract JSON from a plain markdown fence without any language specification.

Why Needed: This test prevents potential issues where the provider may fail to parse JSON from fences with no specified language.

Key Assertions:

- The response is not empty.
- The response starts with ````.
- The response ends with `````.
- The response contains only whitespace characters (spaces, tabs, newlines) between the fence and the start of the JSON.
- The response does not contain any special markdown syntax that might prevent parsing (e.g., `[`, `]`, ```, etc.).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	6 lines (ranges: 38, 42-44, 46-47)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/llm/base.py	20 lines (ranges: 52-53, 186-187, 190-191, 194-195, 198-200, 203, 205, 207, 212, 214-218)
src/pytest_llm_report/llm/schemas.py	7 lines (ranges: 38, 42-43, 50-53)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

📄 tests/test_models.py

29 tests

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

AI ASSESSMENT

Scenario: Tests CoverageEntry serialization correctly.

Why Needed: CoverageEntry serialization may fail if line ranges are not properly formatted.

Key Assertions:

- The 'file_path' key in the dictionary is set to 'src/foo.py'.
- The 'line_ranges' key in the dictionary is set to '1-3, 5, 10-15'.
- The 'line_count' key in the dictionary is set to 10.
- The line ranges are properly formatted (e.g., no overlapping or missing numbers).

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`4 lines (ranges: 260-263)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

PASSED `tests/test_models.py::TestCollectionError::test_to_dict` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the `to_dict` method of `CoverageEntry` class.

Why Needed: This test prevents a potential bug where the `to_dict` method does not correctly serialize the coverage entry data.

Key Assertions:

- The 'file_path' key in the dictionary should be set to 'src/foo.py'.
- The 'line_ranges' key in the dictionary should be set to '1-3, 5, 10-15'.
- The 'line_count' key in the dictionary should be set to 10.
- The values of 'file_path', 'line_ranges', and 'line_count' in the dictionary should match their respective attributes of the `CoverageEntry` class.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	3 lines (ranges: 213-215)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestCoverageEntry::test_to_dict` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that a CoverageEntry object can be serialized correctly into a dictionary.

Why Needed: This test prevents a regression where the coverage entry data is not properly converted to a dictionary.

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: 380-381, 384, 388-390)

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

AI ASSESSMENT

Scenario: Test that an empty annotation has default values.

Why Needed: This test prevents a regression where the annotation is left with no information.

Key Assertions:

- annotation.scenario == "" (empty string)
- annotation.why_needed == "Empty annotation should have default values." (description of why it's necessary)
- annotation.key_assertions == [] (expected empty list for key assertions)
- assert annotation.confidence is None (expected confidence to be None)
- assert annotation.error is None (expected error to be None)

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

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 to a dictionary with the required fields.

Why Needed: This test prevents regression by ensuring that the minimal annotation format is correctly implemented.

Key Assertions:

- The dictionary should contain the keys 'scenario', 'why_needed', and 'key_assertions'.
- The key 'confidence' should not be included in the dictionary when it's None.
- The value of 'scenario' should be present in the dictionary.
- The value of 'why_needed' should be present in the dictionary.
- The value of 'key_assertions' should be present in the dictionary.
- The value of 'confidence' should not be included when it's None.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`8 lines (ranges: 104-107, 109, 111, 113, 115)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

PASSED `tests/test_models.py::TestLlmAnnotation::test_to_dict_with_all_fields` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test to dictionary with all fields

Why Needed: Prevents a potential bug where the full annotation is not included in the dictionary.

Key Assertions:

- Assert that the 'scenario' field contains the expected value.
- Assert that the 'confidence' field contains the expected value.
- Assert that the 'context_summary' field contains the expected key-value pairs.
- Assert that the 'mode' field matches the expected string.
- Assert that all fields are present 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`	10 lines (ranges: 104-107, 109-111, 113-115)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestReportRoot::test_default_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/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: 380-381, 384, 388-390)

PASSED tests/test_models.py::TestReportRoot::test_report_with_collection_errors 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test Report Root::test_report_with_collection_errors verifies that the test reports collection errors correctly.

Why Needed: This test prevents a regression where collection errors are not reported as expected.

Key Assertions:

- The report should contain exactly one collection error.
- The first collection error should be for the file "test_bad.py".
- The node ID of the first collection error should match the provided node ID.

COVERAGE

src/pytest_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: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestReportRoot::test_report_with_warnings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test reports with warnings should be included in the output.

Why Needed: This test prevents a regression where warnings are not reported correctly.

Key Assertions:

- The length of `report.warnings` is 1 and its first element has code `W001`.
- Each warning in `report.warnings` has a 'code' attribute matching the expected value `W001`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	60 lines (ranges: 235-237, 239, 241, 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516-518, 520, 522, 524, 526, 528)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 sorting algorithms used.

Key Assertions:

- The list of nodeids returned by `to_dict()` method should contain all unique nodeids from the input data.
- The list of nodeids returned by `to_dict()` method should have no duplicates.
- All nodeids in the output list should be present in the input data.
- If a test is not found, it should not be included in the sorted list.
- If two tests are not found, they should not be included in the sorted list.
- The order of tests is consistent across different runs of the test suite.
- All nodeids in the output list should be present in the input data even if some tests fail.
- If a test fails, it should still be included in the sorted 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	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: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestReportWarning::test_to_dict_with_detail` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test `test_to_dict_with_detail` verifies that the `to_dict()` method of a `ReportWarning` object returns the correct detail path.

Why Needed: This test prevents a potential issue where the detail path is not correctly extracted from the warning message, potentially leading to incorrect reporting or analysis of warnings.

Key Assertions:

- The value of `d['detail']` should be `/path/to/file`.
- The key `detail` in the dictionary should contain the string `/path/to/file`.
- The detail path is correctly extracted from the warning message.
- The test does not fail when the warning message contains additional details.
- The test does not pass if the warning message does not contain a detail path.
- The `to_dict()` method returns an empty dictionary for invalid warnings.
- The `to_dict()` method raises an exception for invalid 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/models.py	6 lines (ranges: 235-237, 239-241)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestReportWarning::test_to_dict_without_detail` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test to dictionary without detail should exclude it.

Why Needed: Prevents a warning about missing detailed information.

Key Assertions:

- The 'detail' key is expected to be present in the dictionary.
- The value of 'detail' is not provided when it's absent.
- The test ensures that the 'detail' key does not exist in the dictionary.
- The presence or absence of 'detail' affects the warning message.
- Without 'detail', the warning message becomes misleading.

COVERAGE

`src/pytest_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: 380-381, 384, 388-390)`

PASSED `tests/test_models.py::TestRunMeta::test_aggregation_fields_present` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test RunMeta to ensure it has aggregation fields.

Why Needed: To prevent regression where RunMeta is missing or incorrectly configured aggregation fields.

Key Assertions:

- The 'run_id' key should be present in the meta dictionary.
- The 'run_group_id' key should be present in the meta dictionary.
- The 'is_aggregated' key should be set to True.
- The 'aggregation_policy' key should be set to 'merge'.
- The 'run_count' key should be set to 3.
- The length of the 'source_reports' list 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: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestRunMeta::test_llm_fields_excluded_when_disabled` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test LLM fields are excluded when annotations not enabled.

Why Needed: To prevent regression when annotations are disabled, this test ensures that the required fields (llm_annotations_enabled, llm_provider, and llm_model) are missing from the RunMeta object.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	29 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestRunMeta::test_llm_traceability_fields` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test LLM traceability fields are included when enabled.

Why Needed: This test prevents a regression where the 'llm_traceability_fields' configuration is not properly set for LLMs.

Key Assertions:

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

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/models.py`	`40 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413-425)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

PASSED `tests/test_models.py::TestRunMeta::test_non_aggregated_excludes_source_reports` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'Non-aggregated report should not include source_reports' verifies that a non-aggregated run does not include source reports.

Why Needed: This test prevents regression where the 'source_reports' field is included in aggregated runs, potentially misleading users.

Key Assertions:

- The 'source_reports' key is not present in the 'd' dictionary.
- The value of 'is_aggregated' is True for the 'd' dictionary.
- The 'source_reports' key should be absent from the 'd' dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	29 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 a potential bug where the `git_sha` and `git_dirty` fields are not properly initialized in the `RunMeta` object.

Key Assertions:

- The `git_sha` field is set to 'abc1234'.
- The `git_dirty` field is set to True.
- The `repo_version` field is set to '1.0.0'.
- The `repo_git_sha` field is set to 'abc1234'.
- The `repo_git_dirty` field is set to True.
- The `plugin_git_sha` field is set to 'def5678'.
- The `plugin_git_dirty` field is set to False.
- The `config_hash` field is set to 'def5678'.
- The length of the `source_reports` list is 1.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	49 lines (ranges: 283-285, 287-289, 370-386, 388-411, 413, 425)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestRunMeta::test_run_status_fields` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test RunMeta to include run status fields.

Why Needed: This test prevents a potential bug where the 'run_status' field is missing from the RunMeta object.

Key Assertions:

- The 'exit_code' key should be present and equal to 1.
- The 'interrupted' key should be True.
- The 'collect_only' key should be True.
- The 'collected_count' key should be equal to 10.
- The 'selected_count' key should be equal to 8.
- The 'deselected_count' key 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	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: 380-381, 384, 388-390)

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: This test prevents regression where the schema version is not in semver format, potentially causing issues with backward compatibility or compatibility with certain dependencies.

Key Assertions:

- The schema version should be split into three parts (e.g., '1.2.3').
- Each part of the schema version should consist only of digits (e.g., '012', not '123').
- All three parts of the schema version should be present and have a length of 3 characters.
- The first part of the schema version should be greater than or equal to 0.
- The second part of the schema version should be greater than or equal to 0 but less than the first part.
- The third part of the schema version should be greater than or equal to 0 and less than the second part.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestSchemaVersion::test_schema_version_in_repo rt_root` 1ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	54 lines (ranges: 370-386, 388, 391, 393, 396, 399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestSourceCoverageEntry::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:

- The 'file_path' key is set to the expected value.
- The 'line_ranges' key is set to the expected value.
- The 'line_count' key is set to the expected value.
- Each assertion checks that the corresponding field in the dictionary matches the expected value.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	8 lines (ranges: 71-78)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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 the required fields.

Why Needed: This test prevents regression by ensuring that the minimal annotation format is correctly serialized and includes all necessary information.

Key Assertions:

- The 'scenario' field should be present in the dictionary.
- The 'why_needed' field should also be present in the dictionary.
- The 'key_assertions' field should be included in the dictionary, which contains critical checks for confidence.

COVERAGE

src/pytest_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: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestSourceReport::test_to_dict_with_run_id` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The `SourceReport` object should include its `run_id` in the dictionary representation.

Why Needed: This test prevents a potential bug where the `SourceReport` object is not properly serialized with its run ID.

Key Assertions:

- The `run_id` key is present and has the value 'run-1'.
- The `sha256` key is also present and contains the correct value 'abc123'.
- The `path` key is present and contains the expected string '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/models.py	6 lines (ranges: 283-285, 287-289)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestSummary::test_to_dict` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `CoverageEntry` object can be serialized correctly into a dictionary.

Why Needed: This test prevents a potential bug where the serialization of `CoverageEntry` objects could result in incorrect data being stored or transmitted.

Key Assertions:

- The expected value for `file_path` is 'src/foo.py'.
- The expected value for `line_ranges` is '1-3, 5, 10-15'.
- The expected value for `line_count` is 10.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	11 lines (ranges: 455-463, 465, 467)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestTestCaseResult::test_minimal_result` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that a minimal result has the required fields.

Why Needed: This test prevents regression by ensuring that a minimal result always contains all necessary information.

Key Assertions:

- The 'nodeid' field is set to the expected value.
- The 'outcome' field is set to 'passed'.
- The 'duration' field is set to 0.0 (indicating no execution time).
- The 'phase' field is set to 'call'.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/models.py`	19 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194, 196)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestTestCaseResult::test_result_with_coverage` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: tests/test_models.py::TestTestCaseResult::test_result_with_coverage verifies that the 'result' dictionary contains a single 'coverage' key with a list of coverage entries.

Why Needed: This test prevents regression by ensuring that the 'result' dictionary always has a single 'coverage' key and includes all required information about the coverage.

Key Assertions:

- The 'coverage' key in the 'result' dictionary should be present.
- The 'coverage' key in the 'result' dictionary should have exactly one entry.
- The 'file_path' of the first 'coverage' entry in the 'result' dictionary should match the expected file path.
- All required information about the coverage (line ranges and line count) should be included in the 'coverage' 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`	`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: 380-381, 384, 388-390)`

PASSED tests/test_models.py::TestTestCaseResult::test_result_with_llm_opt_out 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the `TestCaseResult` class with LLM opt-out.

Why Needed: Prevents regression in case where LLM is opted out and the test fails.

Key Assertions:

- The 'llm_opt_out' key should be present in the result dictionary.
- The value of 'llm_opt_out' should be `True` when `llm_opt_out=True`.
- If `llm_opt_out=False`, the 'llm_opt_out' key should not be present in the result dictionary.
- If `llm_opt_out=False` and the test passes, the 'llm_opt_out' value should remain `False`.
- If `llm_opt_out=False` and the test fails, the 'llm_opt_out' value should be `True`.
- The 'llm_opt_out' key should not be present in the result dictionary if it is set to `None` or an empty string.
- The 'llm_opt_out' key should have a consistent value across all tests with LLM opt-out.
- The test should fail when `llm_opt_out=False` and the 'llm_opt_out' key is not present in the result dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: 380-381, 384, 388-390)

PASSED `tests/test_models.py::TestTestCaseResult::test_result_with_rerun` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test verifies that the `result` dictionary contains the expected 'rerun_count' and 'final_outcome' values.

Why Needed: This test prevents a regression where the result is not correctly updated with rerun counts or final outcomes when running tests in parallel.

Key Assertions:

- The value of `d['rerun_count']` should be equal to 2.
- The value of `d['final_outcome']` should be equal to 'passed'.
- The presence of `rerun_count` and `final_outcome` keys in the dictionary is expected.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	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: 380-381, 384, 388-390)

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 `TestCaseResult` object does not include 'rerun_count' and 'final_outcome' fields.

Why Needed: This test prevents a regression where the `TestCaseResult` object includes these fields, potentially misleading users about the result of the test.

Key Assertions:

- The 'rerun_count' field is not present in the `TestCaseResult` object.
- The 'final_outcome' field is not present in the `TestCaseResult` object.
- The presence of these fields could lead to incorrect interpretation of the result by users.

COVERAGE

`src/pytest_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: 380-381, 384, 388-390)

📄 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 for the test_default_values scenario.

Why Needed: This test prevents a potential regression where the default values of Config are not set correctly, potentially leading to unexpected behavior in subsequent tests or operations.

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: 107, 147, 224)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options.py::TestConfig::test_get_default_config` 1ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	3 lines (ranges: 107, 147, 233)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options.py::TestConfig::test_is_llm_enabled` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the `is_llm_enabled` check returns False when the provider is 'none', and True when it's 'ollama'.

Why Needed: Prevents a regression where the test fails due to an incorrect implementation of the `is_llm_enabled` method.

Key Assertions:

- The `is_llm_enabled()` method should return False for a provider with no configuration.
- The `is_llm_enabled()` method should return True for a provider with 'ollama' as its configuration.
- A new instance of the `Config` class without any configuration should also return False.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	3 lines (ranges: 107, 147, 224)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-197, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED tests/test_options.py::TestConfig::test_validate_invalid_context_mode 1ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-189, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED tests/test_options.py::TestConfig::test_validate_invalid_include_phase 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify the test validates an invalid include phase.

Why Needed: Prevent a potential bug where an invalid include phase is used.

Key Assertions:

- The configuration object `cfg` has a single error with message 'Invalid include_phase 'lunch_break'.
- The error message contains the string 'lunch_break'.
- The test asserts that there is exactly one error in the list of errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-205, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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	19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 due to invalid configuration values that could lead to unexpected behavior.

Key Assertions:

- cfg.validate() should return at least 5 error messages.
- The 'llm_context_bytes' value should be at least 1000.
- The 'llm_max_tests' value should be either 0 or positive.
- The 'llm_requests_per_minute' value should be at least 1.
- The 'llm_timeout_seconds' value should be at least 1.
- The 'llm_max_retries' value should be either 0 or positive.
- All error messages should contain the specified constraint messages.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	22 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209-218, 220)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options.py::TestConfig::test_validate_valid_config` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Testing the `validate` method of the `Config` class with a valid configuration.

Why Needed: This test prevents potential bugs or regressions that could occur when validating a valid configuration.

Key Assertions:

- A `Config` object is created and assigned to the `cfg` variable.
- The `validate()` method of the `Config` class is called and returns an empty list (`[]`).
- An assertion is made that `errors` is `None`, indicating no validation errors were found.
- The configuration is successfully validated without any issues or errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	17 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED tests/test_options.py::TestLoadConfig::test_load_aggregation_options 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test loads aggregation options for pytest configuration.

Why Needed: Prevents regression in aggregation option loading when using merge policy.

Key Assertions:

- The aggregate_dir attribute is set to 'aggr_dir'.
- The aggregate_policy attribute is set to 'merge'.
- The aggregate_run_id attribute is set to 'run-123'.
- The aggregate_group_id attribute is set to 'group-abc'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	28 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286-294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options.py::TestLoadConfig::test_load_config_invalid_int_ini` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test Load Config with Invalid Integer Value in INI File

Why Needed: This test prevents a potential crash when encountering invalid integer values in the INI configuration file.

Key Assertions:

- The function `load_config` should not crash or throw an exception when it encounters an invalid integer value in the INI configuration file.
- The default value of `llm_max_retries` should be used instead of the provided invalid value 'garbage'.
- The test should pass even if the `getini` method throws a `ValueError` exception for the specified key.
- The function `load_config` should not raise an error when it encounters an invalid integer value in the INI configuration file, but instead return the default value or fallback to a reasonable alternative.
- The test should verify that the `llm_max_retries` value is set correctly after the crash or fallback occurs.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	28 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263-267, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options.py::TestLoadConfig::test_load_coverage_source` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `llm_coverage_source` option is set to 'cov_dir' when loaded.

Why Needed: This test prevents a potential bug where the coverage source is not correctly set, potentially leading to incorrect coverage analysis.

Key Assertions:

- cfg.llm_coverage_source should be set to 'cov_dir'
- cfg.llm_coverage_source is equal to 'cov_dir'
- The value of `llm_coverage_source` in the configuration file matches the expected value
- The test verifies that the coverage source option is correctly set
- The test checks for any potential errors or exceptions during configuration loading

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	25 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options.py::TestLoadConfig::test_load_defaults` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the default provider is 'none' and report HTML is None when no options are set.

Why Needed: Prevents a potential bug where the test fails with an incorrect provider or report settings.

Key Assertions:

- The `cfg.provider` attribute should be set to 'none'.
- The `cfg.report_html` attribute should be set to None.
- The default provider is correctly identified as 'none'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	24 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 bug where the CLI options override the ini options, potentially causing unexpected behavior or incorrect results.

Key Assertions:

- CLI values should win over ini values for html report.
- CLI values should set llm_requests_per_minute to 100.
- ini values should not be overridden by CLI options.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	27 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259-261, 263, 270-272, 274, 276, 278, 280-282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options.py::TestLoadConfig::test_load_from_cli_retries` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `llm_max_retries` option is correctly set to 2 when loading configuration from CLI.

Why Needed: This test prevents a potential bug where the `llm_max_retries` option is not properly set, causing unexpected behavior in the application.

Key Assertions:

- The value of `llm_max_retries` should be equal to 2 when the option is set.
- The configuration object should have an attribute `llm_max_retries` with a value of 2.
- The `load_config` function should return a configuration object with an `llm_max_retries` attribute equal to 2.
- The `assert` statement should raise an AssertionError if the `llm_max_retries` option is not set or has a different value.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	25 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282-283, 286, 288, 290, 292, 294, 298, 300)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options.py::TestLoadConfig::test_load_from_ini` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test loading values from ini options.

Why Needed: Prevents a potential bug where the test fails if the 'llm_report_provider' option is not set in the ini file.

Key Assertions:

- The 'provider' key in the config should be equal to 'ollama'.
- The 'model' key in the config should be equal to 'llama3'.
- The 'context_mode' key in the config should be equal to 'balanced'.
- The 'requests_per_minute' key in the config should be equal to 10.
- The 'max_retries' key in the config should be equal to 2.
- The 'report_html' key in the config should be equal to 'report.html'.
- The 'report_json' key in the config should be equal to 'report.json'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`32 lines (ranges: 107, 147, 248, 251-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

📄 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 policy is set to "merge" instead of "append", causing incorrect aggregation results.

Key Assertions:

- The `aggregate_dir` attribute should be set to `/reports`.
- The `aggregate_policy` attribute should be set to "merge".
- The `aggregate_include_history` attribute should be set to `True`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_all_output_paths` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test Config with all output paths.

Why Needed: Prevents a potential bug where the test fails if the report format is changed without updating the configuration.

Key Assertions:

- The `report_html` attribute should be equal to 'report.html'.
- The `report_json` attribute should be equal to 'report.json'.
- The `report_pdf` attribute should be equal to 'report.pdf'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_capture_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the configuration of capturing failed output and setting a maximum number of characters for the captured output.

Why Needed: This test prevents a potential issue where the captured output exceeds the specified maximum length, potentially causing unexpected behavior or errors.

Key Assertions:

- config.capture_failed_output is set to True
- config.capture_output_max_chars is set to 8000

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_compliance_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `Config` object is correctly initialized with compliance settings.

Why Needed: This test prevents a bug where the `Config` object's metadata and HMAC key files are not properly set.

Key Assertions:

- The `metadata_file` attribute of the `Config` object is set to 'metadata.json'.
- The `hmac_key_file` attribute of the `Config` object is set to 'key.txt'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_coverage_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test configuration with coverage settings.

Why Needed: Prevents a potential bug where the test configuration does not include all phases for better coverage.

Key Assertions:

- config.omit_tests_from_coverage is False
- config.include_phase == "all"

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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` patterns should be included in the `llm_context_exclude_globs` list.
- Custom exclude globs should override or be added to the default exclude globs if present.
- The test should fail when custom exclude globs are provided without a corresponding configuration value.
- The `llm_context_exclude_globs` list should contain all the specified patterns.
- No additional patterns should be included in the `llm_context_exclude_globs` list for files that do not match any of the specified patterns.
- Custom exclude globs should have priority over default ones if both are provided.
- The test should pass when custom exclude globs are properly configured and matched against the file paths.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	1 lines (ranges: 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED tests/test_options_extended.py::TestConfigAnnotations::test_include_globs 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `llm_context_include_globs` option includes only `.py` files.

Why Needed: This test prevents a potential bug where the `llm_context_include_globs` option is not correctly applied, causing incorrect file inclusion.

Key Assertions:

- The `*.py` glob matches only files with a `.py` extension.
- The `*.pyi` glob matches only files with a `.pyi` extension.
- The `llm_context_include_globs` option is correctly applied to the configuration.
- The included files are not empty or contain any non-ASCII characters.
- No other files than those specified in the `llm_context_include_globs` option are included.
- The file inclusion matches the expected pattern for the given glob patterns.
- The test does not fail when including a single `.pyi` file with an empty contents.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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: 107)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_llm_execution_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify the correctness of LLM execution settings configuration.

Why Needed: Prevents a potential bug where the maximum number of tests is not set correctly, potentially leading to unexpected behavior or errors in test execution.

Key Assertions:

- The value of `llm_max_tests` is indeed 50.
- The value of `llm_max_concurrency` is indeed 8.
- The value of `llm_requests_per_minute` is indeed 12.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_llm_param_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Configures the LLM with param settings and verifies their values.

Why Needed: Prevents a potential bug where the LLM's parameter value exceeds the maximum allowed length of 200 characters.

Key Assertions:

- config.llm_include_param_values is set to True
- assert config.llm_param_value_max_chars == 200

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_llm_settings` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the configuration with LLM settings.

Why Needed: Prevents regression in LLM settings configuration.

Key Assertions:

- The provider is set to `ollama`.
- The model is set to `llama3.2`.
- The context bytes are set to 64KB.
- The context file limit is set to 20 files.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_repo_root_path` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that the `repo_root` attribute of the test configuration is set to `/project`.

Why Needed: This test prevents a potential bug where the test configuration does not have a specified repo root, causing issues with repository setup.

Key Assertions:

- The `repo_root` attribute of the test configuration should be equal to `Path('/project')`.
- The `repo_root` attribute of the test configuration should be a valid path.
- The test configuration should have a specified repo root.
- The test configuration should not inherit its repo root from another parent configuration.
- The test configuration's repo root should not be empty or None.
- The test configuration's repo root should be a directory (not a file).
- The test configuration's repo root should be accessible via the file system.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_extended.py::TestConfigAnnotations::test_valid_phase_values` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that all valid include_phase values pass validation.

Why Needed: Prevents regression of invalid phase values being included in the configuration.

Key Assertions:

- A Config instance with `include_phase` set to 'run' should not raise any validation errors.
- A Config instance with `include_phase` set to 'setup' should not raise any validation errors.
- A Config instance with `include_phase` set to 'teardown' should not raise any validation errors.
- A Config instance with `include_phase` set to 'all' should not raise any validation errors.
- A Config instance with an invalid include_phase value (e.g. 'invalid') should raise a ValueError.
- The validate method of the Config instance should return an empty list of errors for 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	17 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

📄 tests/test_options_maximal.py

10 tests

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_exclude_globs` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the default exclude globs for the LLM context.

Why Needed: This test prevents a potential bug where the default exclude globs are not correctly set.

Key Assertions:

- The function `defaults` should contain the glob pattern `*.pyc`.
- The function `defaults` should contain the glob pattern `__pycache__/*`.
- The function `defaults` should contain the glob pattern `*secret*`.
- The function `defaults` should contain the glob pattern `*password*`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_redact_patterns` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test default redact patterns with various flags.

Why Needed: Prevents a potential bug where the test fails due to missing or incorrect pattern matching.

Key Assertions:

- The '--password' flag should be present in any correct pattern.
- The '--token' flag should be present in any correct pattern.
- Any pattern containing '--api[_-]?key' should include this flag.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_maximal.py::TestConfigDefaultsMaximal::test_default_values` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the default values of the configuration.

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

Key Assertions:

- config.provider should be 'none' by default.
- config.llm_context_mode should be 'minimal' by default.
- config.llm_context_bytes should be 32000 bytes by default.
- config.omit_tests_from_coverage should always be True.
- config.include_phase should always be 'run' for the default 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`	`3 lines (ranges: 107, 147, 233)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

PASSED `tests/test_options_maximal.py::TestConfigHelpersMaximal::test_is_llm_enabled` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies whether the LLM is enabled based on the provider.

Why Needed: Prevents regression in LLM detection when switching providers.

Key Assertions:

- The function should return False for a non-existent provider.
- The function should return True for 'ollama' and 'litellm' providers.
- The function should return True for the 'gemini' provider.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	3 lines (ranges: 107, 147, 224)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_aggregate_policy` 1ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-197, 201-202, 209, 211, 213, 215, 217, 220)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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	20 lines (ranges: 107, 147, 175, 178-179, 185-189, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_include_phase` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test validates the maximum include phase.

Why Needed: Prevents a potential bug where an invalid include phase is used, causing the validation to fail and potentially leading to unexpected behavior.

Key Assertions:

- The test verifies that the `validate()` method returns exactly one error for an invalid include phase.
- The error message contains the string 'Invalid include_phase 'invalid'
- The error message includes the specified include phase value.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	20 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-205, 209, 211, 213, 215, 217, 220)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_invalid_provider` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test validates an invalid provider.

Why Needed: Prevents a potential bug where the test fails due to an invalid provider being used.

Key Assertions:

- The function `validate()` should return exactly one error message for an invalid provider.
- The error message should contain the string 'Invalid provider ' followed by the actual invalid provider name.
- The error message should be a single line of text containing the invalid provider name.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	19 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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, llm_max_tests, llm_requests_per_minute, llm_timeout_seconds are set to invalid numeric values.

Key Assertions:

- The validate method should return an error for invalid numeric values of llm_context_bytes.
- The validate method should return an error for invalid numeric values of llm_max_tests.
- The validate method should return an error for invalid numeric values of llm_requests_per_minute.
- The validate method should return an error for invalid numeric values of llm_timeout_seconds.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	21 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209-217, 220)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_options_maximal.py::TestConfigValidationMaximal::test_validate_valid_config` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `validate()` method of a valid configuration returns an empty list.

Why Needed: Prevents potential issues with invalid or malformed configurations where no validation is performed.

Key Assertions:

- The `validate()` method should return an empty list for a valid configuration.
- A valid configuration should not raise any exceptions during validation.
- No errors or warnings should be raised when validating a valid configuration.
- The `Config` object's `validate()` method should not throw an exception if the input is invalid.
- The `validate()` method should only return an empty list for valid configurations, without raising any other 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/options.py`	17 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

📄 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 `load_config` function returns a `Config` instance with default settings.

Why Needed: This test prevents potential bugs where the plugin configuration is missing or incomplete due to lack of default settings.

Key Assertions:

- The `cfg` variable should be an instance of `Config`.
- The `cfg` variable should have some default values set.
- If `pytestconfig` does not have a valid options section, pytest may not be able to load the configuration 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	33 lines (ranges: 107, 147, 248, 251-259, 261, 263-265, 270, 272-276, 278, 280, 282, 286, 288, 290-292, 294, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_plugin_integration.py::TestPluginConfigLoading::test_markers_exist_in_config` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Verify that the `pytestconfig` object is accessible.

Why Needed: Prevent a potential bug where the configuration markers are missing from the test environment.

Key Assertions:

- The `pytestconfig` object should be defined and not None.
- The `pytestconfig` object should have attributes that match its expected structure (e.g. `markers`, `groups`, etc.).
- Any configuration markers or groups should be accessible through the `pytestconfig` object.
- The `pytestconfig` object should be able to be used as a dictionary-like object for further processing or manipulation.
- The `pytestconfig` object should not raise any exceptions when accessed or modified.
- The `pytestconfig` object's attributes and methods should match its expected behavior (e.g. `add_marker()`, `remove_marker()`, etc.).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_plugin_integration.py::TestPluginIntegration::test_llm_context_marker` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: The test verifies that the context marker does not cause errors.

Why Needed: This test prevents regression in the LLM integration plugin, ensuring that the context marker does not interfere with test execution.

Key Assertions:

- Asserts that the context marker is not causing any errors or exceptions during test execution.
- Verifies that the context marker is properly configured and enabled in the plugin configuration.
- Checks for any potential issues with the context marker, such as incorrect usage or compatibility problems.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

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: 380-381, 384, 388-390)

PASSED tests/test_plugin_integration.py::TestPluginIntegration::test_requirement_marker 1ms 🛡️ 2

AI ASSESSMENT

Scenario: The `requirement_marker` function is being tested to ensure it does not throw any errors.

Why Needed: This test prevents a potential bug where the `requirement_marker` function might cause unexpected behavior or errors.

Key Assertions:

- The `requirement_marker` function should be called without raising an exception.
- Any exceptions raised by the `requirement_marker` function should be caught and handled properly.
- The `requirement_marker` function should not modify any external state in a way that could cause unexpected behavior.
- Any side effects of the `requirement_marker` function should be thoroughly tested to ensure they do not impact other parts of the codebase.
- The `requirement_marker` function should adhere to all specified requirements and constraints.
- Any assumptions made about the `requirement_marker` function's behavior based on its documentation or usage should be validated through testing.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

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 module.

Why Needed: This test prevents a regression where the report generation process fails to produce a JSON file or HTML document.

Key Assertions:

- The total number of tests passed should be 2.
- The number of tests that passed should be 1.
- The text 'test_a.py' should be found in the generated HTML document.
- The text 'test_b.py' should be found in the generated HTML document.

COVERAGE

src/pytest_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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 disabled and the plugin is not enabled.

Why Needed: To prevent a regression where the plugin's collect report functionality is disabled but still functional.

Key Assertions:

- The `pytest_collectreport` function should be called with `_enabled_key` set to 'collectreport' and `False` as its argument when the session's stash returns False.
- The `session.config.stash.get` method should return a mock object that matches the expected behavior.
- The `mock_report.session.config.stash.get.return_value` attribute should be set to `False` during the test.
- The `pytest_collectreport` function should not be called with `_enabled_key` set to 'collectreport' and `True` as its argument when the session's stash returns False.
- The `mock_report.session.config.stash.get.assert_called_with(_enabled_key, False)` assertion should pass during the test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	10 lines (ranges: 380-381, 384, 388-390, 401-402, 408-409)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_enabled` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that collectreport calls collector when enable is True.

Why Needed: To prevent a potential bug where the plugin does not call the collector when it should, and instead returns an empty report.

Key Assertions:

- The `pytest_collectreport` function is called with the correct mock report instance.
- The `mock_collector.handle_collection_report` method is called once with the correct mock report instance.
- The `stash_get` function returns True for `_enabled_key` and `True` for `_collector_key` when the plugin is enabled.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	12 lines (ranges: 380-381, 384, 388-390, 401-402, 408, 412-414)

PASSED `tests/test_plugin_maximal.py::TestPluginCollectReport::test_pytest_collectreport_no_session` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Verify that `pytest_collectreport` does not throw an exception when no session is available.

Why Needed: Prevent a potential bug where the plugin throws an exception due to missing session attribute.

Key Assertions:

- mock_report.session is deleted before pytest_collectreport is called
- pytest_collectreport() should not raise an exception
- the function does not throw any errors or exceptions
- the function behaves as expected without raising an exception
- the plugin does not crash or terminate unexpectedly

COVERAGE

`src/pytest_llm_report/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: 380-381, 384, 388-390, 401, 405)

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: Prevent regression in case a plugin's `session` attribute is not properly initialized or is set to `None` unexpectedly.

Key Assertions:

- The function `pytest_collectreport` should not throw any exceptions when called with a `None` session.
- A `None` session should be passed without raising an exception.
- No error message or indication of failure should be displayed when calling `pytest_collectreport` with 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: 380-381, 384, 388-390, 401, 405)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_llm_enabled_warning` 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that LLM enabled warning is raised when using the Ollama LLM report provider.

Why Needed: The test prevents a potential bug where the LLM provider 'ollama' is enabled without being configured correctly, resulting in an unhandled warning.

Key Assertions:

- mock_config.option.llm_report_provider should be set to 'ollama'.
- mock_config.getini should return the expected values for llm_report_provider, llm_report_model, llm_report_context_mode.
- mock_config.option.llm_report_html should not be set to None.
- mock_config.option.llm_report_json should not be set to None.
- mock_config.option.llm_report_pdf should not be set to None.
- mock_config.option.llm_evidence_bundle should not be set to None.
- mock_config.option.llm_dependency_snapshot should not be set to None.
- mock_config.option.llm_requests_per_minute should not be set to None.
- mock_config.option.llm_aggregate_dir should not be set to None.
- mock_config.option.llm_aggregate_policy should not be set to None.
- mock_config.option.llm_aggregate_run_id should not be set to None.
- mock_config.option.llm_aggregate_group_id should not be set to None.
- mock_config.rootpath should be set to the expected path.
- mock_config.stash should be an empty dictionary.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	44 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	29 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 380-381, 384, 388-390)

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: Prevents a potential bug where the plugin does not handle invalid configuration properly and raises an exception instead of providing informative error messages.

Key Assertions:

- The `pytest_configure` function is called with a mock `mock_config` object that has been configured to raise a `UsageError` when validation errors occur.
- The `mock_config.option.llm_report_html`, `option.llm_report_json`, `option.llm_report_pdf`, `option.llm_evidence_bundle`, `option.llm_dependency_snapshot`, `option.llm_requests_per_minute`, `option.llm_aggregate_dir`, `option.llm_aggregate_policy`, `option.llm_aggregate_run_id`, and `option.llm_aggregate_group_id` are all set to `None` in the mock configuration.
- The `mock_config.rootpath` is set to `/project` which is a valid path, but the `stash` dictionary is empty.
- The `pytest_configure` function is called with the invalid configuration provided through the `ini_values` dictionary.
- The `UsageError` exception is raised with a message that indicates there were configuration errors.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	43 lines (ranges: 107, 147, 175, 178-181, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 248, 251-253, 255, 257, 259, 261, 263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	25 lines (ranges: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-199, 380-381, 384, 388-390)

PASSED `tests/test_plugin_maximal.py::TestPluginConfigure::test_pytest_configure_worker_skip` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that configure skips on xdist workers.

Why Needed: This test prevents a regression where the plugin might skip configuration due to an issue with the xdist worker.

Key Assertions:

- The `pytest_configure` function should return early without calling `addinivalue_line`.
- The `addinivalue_line` method is still called for markers before the worker check.
- The `workerinput` attribute of the mock configuration object is not updated correctly.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	17 lines (ranges: 169-171, 173-175, 177-179, 183-184, 380-381, 384, 388-390)

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: 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 380-381, 384, 388-390)

PASSED tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_all_ini_options 2ms 🛡️ 3

AI ASSESSMENT

Scenario: Test loading all INI options for the plugin.

Why Needed: This test prevents a potential issue where the plugin does not load configuration from INIs, potentially causing unexpected behavior or errors.

Key Assertions:

- The `llm_report_provider` option is set to 'ollama'.
- The `llm_report_model` option is set to 'llama3.2'.
- The `llm_report_context_mode` option is set to 'complete'.
- The `llm_report_requests_per_minute` option is set to 10.
- The `report_html` option is set to 'ini.html'.
- The `report_json` option is set to 'ini.json'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	31 lines (ranges: 107, 147, 248, 251-263, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_plugin_maximal.py::TestPluginLoadConfig::test_load_config_cli_overrides_ini` 2ms 🛡️ 3

AI ASSESSMENT

Scenario: Test CLI options override INI options.

Why Needed: This test prevents regression where CLI options override INI options, ensuring the correct output is displayed in the report.

Key Assertions:

- The `report_html` option should be set to 'cli.html'.
- The `report_json` option should be set to 'cli.json'.
- The `report_pdf` option should be set to 'cli.pdf'.
- The `report_evidence_bundle` option should be set to 'bundle.zip'.
- The `report_dependency_snapshot` option should be set to 'deps.json'.
- The value of `llm_requests_per_minute` should not exceed 20.
- The value of `aggregate_dir` should match '/agg'.
- The value of `aggregate_policy` should be 'merge'.
- The value of `aggregate_run_id` should match 'run-123'.
- The value of `aggregate_group_id` should match 'group-abc'.
- The root path 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	38 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259-263, 270-283, 286-295, 298, 300)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::test_terminal_summary_disabled` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that terminal summary skips when plugin is disabled.

Why Needed: Prevents regression where terminal summary is not displayed due to a disabled plugin.

Key Assertions:

- Mocked stash.get was called with _enabled_key as argument and False value.
- Mocked stash.get was called once with _enabled_key as argument and False value.
- Mocked stash.get was called twice with _enabled_key as argument, but the second call did not return a result.
- Mocked stash.get was called before the first assertion in test_terminal_summary_disabled function.
- Mocked stash.get was called after the first assertion in test_terminal_summary_disabled function.
- Mocked stash.get was called within the same scope where pytest_terminal_summary is called.
- Mocked stash.get was called with _enabled_key as argument and False value, but it should have been mocked to return None instead.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`9 lines (ranges: 238, 242-243, 380-381, 384, 388-390)`

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::test_terminal_summary_worker_skip` 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`	`8 lines (ranges: 238-239, 380-381, 384, 388-390)`

PASSED `tests/test_plugin_maximal.py::TestPluginMaximal::testload_config` 3ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that config loading from pytest objects (CLI + INI) works correctly.

Why Needed: This test prevents a potential bug where the plugin does not load configuration files from pytest objects.

Key Assertions:

- The `report_html` option is set to 'out.html' when loaded from the mock config.
- The `rootpath` option is set to '/root' when loaded from the mock config.
- The `getini` method of the mock config returns None for all keys instead of raising an exception or returning a default value.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/options.py`	`36 lines (ranges: 107, 147, 248, 251, 253, 255, 257, 259, 261, 263, 270-283, 286-295, 298, 300)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`

PASSED `tests/test_plugin_maximal.py::TestPluginRuntest::test_runtest_makereport_disabled` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test makereport skips when disabled.

Why Needed: Prevents test from failing due to makereport being skipped when the plugin is disabled.

Key Assertions:

- The `pytest_runtest_makereport` hook wrapper should not be called with a mock item that has its stash set to False.
- The `mock_outcome.get_result()` call should return a result from the mock outcome, which is then passed to the next point in the test sequence.
- The `next(gen)` line should yield the generated report without raising an exception.
- The `try` block around the `gen.send(mock_outcome)` call should be executed before the `StopIteration` exception is raised.
- After calling `gen.send(mock_outcome)`, the next point in the test sequence (`mock_call`) should not be called.
- The `next(gen)` line after sending the result should yield the generated report without raising an exception.
- The final call to `try` block around the `gen.send(mock_outcome)` call should complete successfully.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	7 lines (ranges: 380-381, 384-385, 388-390)

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` instance as its second argument.
- The `mock_collector.handle_runtest_logreport` method should be called with the `mock_report` instance and `mock_item` instance as arguments.
- The `mock_collector` instance should have a `handle_runtest_logreport` method that takes two arguments: `mock_report` and `mock_item`.
- The `mock_collector.handle_runtest_logreport` method should not be called with any arguments when the `mock_report` instance is not available.
- The `mock_collector.handle_runtest_logreport` method should not raise an exception when the `mock_report` instance is not available.
- The `mock_collector.handle_runtest_logreport` method should call the `get_result` method on the `mock_report` instance with no arguments.
- The `mock_collector.handle_runtest_logreport` method should not call any other methods on the `mock_collector` instance when it is called without an argument.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_disabled` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that collection_finish is skipped when disabled for Pytest.

Why Needed: This test prevents a regression where collection_finish fails to set the stash key correctly when it's disabled.

Key Assertions:

- mock_session.config.stash.get.assert_called_with(_enabled_key, False)
- pytest_collection_finish(mock_session) returns None
- mock_session.config.stash.get.return_value 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	8 lines (ranges: 380-381, 384, 388-390, 424-425)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_collection_finish_enabled` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that collection_finish calls collector when enabled.

Why Needed: Prevents a potential bug where the plugin's hooks are not called when pytest_collection_finish is enabled.

Key Assertions:

- The stash_get function returns True for _enabled_key and mock_collector.
- The stash_get function returns False for _collector_key.
- The stash_get function calls mock_collector.handle_collection_finish once with the correct arguments.
- The pytest_collection_finish function is called with the correct items.
- The collector's handle_collection_finish method is called with the correct collection of items.
- The collector's handle_collection_finish method does not return any value.
- The stash_get function returns False for _collector_key and mock_collector.
- The stash_get function calls mock_collector.handle_collection_finish once with the correct 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`	10 lines (ranges: 380-381, 384, 388-390, 424, 428-430)

PASSED `tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_disabled` 1ms 🛡️ 2

AI ASSESSMENT

Scenario: Test that session start is skipped when disabled in Pytest.

Why Needed: This test prevents a regression where the plugin's hooks are not executed when pytest_sessionstart is disabled.

Key Assertions:

- mocked config.stash.get.call_count should be 1 (i.e., it was called once)
- mocked _enabled_key call should have been made with False as its argument
- pytest_sessionstart mock object should not have been called
- mock_session.config.stash.get.assert_called_with(_enabled_key, False) should be True

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	8 lines (ranges: 380-381, 384, 388-390, 441-442)

PASSED tests/test_plugin_maximal.py::TestPluginSessionHooks::test_pytest_sessionstart_enabled 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that sessionstart initializes collector when enabled.

Why Needed: Prevents a potential bug where the collector is not created due to an incorrect stash configuration.

Key Assertions:

- The `_collector_key` should be present in `mock_stash`.
- The `_start_time_key` should be present in `mock_stash`.
- A mock session with a configured stash that supports both get() and [] operations should have the collector created.
- _collector_key is not present in mock_stash
- _start_time_key is not present 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: 107, 147)
src/pytest_llm_report/plugin.py	11 lines (ranges: 380-381, 384, 388-390, 441, 445, 448, 450-451)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: The test verifies that pytest_addoption adds the expected arguments to the parser.

Why Needed: This test prevents a potential bug where pytest_addoption does not add the required arguments, causing the plugin to fail or behave unexpectedly.

Key Assertions:

- parser.getgroup.assert_called_with('llm-report', 'LLM-enhanced test reports')
- group.addoption.call_args_list[0][0] == '--llm-report'
- group.addoption.call_args_list[1][0] == '--llm-coverage-source'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	99 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 380-381, 384, 388-390)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_pytest_addoption_ini` 2ms 🛡️ 2

AI ASSESSMENT

Scenario: Verifies pytest_addoption adds INI options (lines 13-34) for the 'llm_report' plugin.

Why Needed: This test prevents a regression where pytest_addoption does not add the expected INI options for the 'llm_report' plugin.

Key Assertions:

- Verify that 'llm_report_html', 'llm_report_json', and 'llm_report_max_retries' are added to the ini file.
- Verify that these options are included in the list of arguments returned by parser.addini.call_args_list.
- Ensure that the correct values ('llm_report_html', 'llm_report_json', and 'llm_report_max_retries') are present in the ini calls.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`99 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 380-381, 384, 388-390)`

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_coverage_calculation` 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: 107, 147, 224)
`src/pytest_llm_report/plugin.py`	53 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-305, 307, 309-312, 324-325, 330-331, 358-368, 380-381, 384, 388-390)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_llm_enabled` 3ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

`src/pytest_llm_report/collector.py`	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
`src/pytest_llm_report/options.py`	3 lines (ranges: 107, 147, 224)
`src/pytest_llm_report/plugin.py`	59 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324-325, 330-333, 336, 338, 341-343, 350-355, 358-368, 380-381, 384, 388-390)

PASSED `tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_no_collector` 2ms 🛡️ 3

AI ASSESSMENT

Scenario: Test terminal summary creates collector if missing.

Why Needed: This test prevents a bug where the plugin does not create a collector for terminals without a stash.

Key Assertions:

- mock_terminalreporter is called with correct arguments.
- stash._enabled_key is True.
- stash._config_key is cfg.
- mock_config.stash is set to stash.
- mock_writer_cls.return_value is mock_writer.
- mock_mapper_cls.return_value is mock_mapper.
- mock_mapper.map_coverage.return_value is an empty dictionary.
- pytest_terminal_summary.mock_terminalreporter.call_count is 1.

COVERAGE

src/pytest_llm_report/collector.py	16 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210, 277, 285)
src/pytest_llm_report/options.py	3 lines (ranges: 107, 147, 224)
src/pytest_llm_report/plugin.py	45 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390)

PASSED tests/test_plugin_maximal.py::TestPluginTerminalSummary::test_terminal_summary_with_aggregation 2ms 🛡️ 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	21 lines (ranges: 238, 242, 246, 249-250, 252-253, 256-257, 259, 261-265, 380-381, 384, 388-390)

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: 107, 147, 224)
src/pytest_llm_report/plugin.py	52 lines (ranges: 238, 242, 246, 249, 268-269, 271, 273, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-305, 315-318, 324-325, 330-331, 358-368, 380-381, 384, 388-390)

📄 tests/test_prompts.py

6 tests

PASSED `tests/test_prompts.py::TestContextAssembler::test_assemble_balanced_context` 9ms 🛡️ 4

AI ASSESSMENT

Scenario: Test Assembly of Balanced Context for `test_a.py::test_1`

Why Needed: Prevents a potential bug where the test is not able to find the required dependencies (utils.py) when assembling the balanced context.

Key Assertions:

- The 'utils.py' file should be present in the assembled context.
- The function `util()` from the 'utils.py' file should be found in the assembled context.
- The line ranges and line count of the coverage entry should match the actual lines in 'utils.py'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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: The `ContextAssembler` is able to assemble a complete context for the test file "test_a.py".

Why Needed: This test prevents regression if the `ContextAssembler` fails to correctly assemble a complete context for the provided test file.

Key Assertions:

- The presence of 'test_1' in the source code is asserted.
- The `source` variable contains the assembled context.
- The `context` variable is not empty.
- The `source` and `context` variables are both strings.
- The `source` string does not contain any additional lines beyond 'def test_1(): pass'.
- The `source` string starts with a comment line that indicates it's part of the test file.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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: The test verifies that the ContextAssembler can assemble a minimal context for a test file with a single test function.

Why Needed: This test prevents regression if there are changes to the `ContextAssembler` or its dependencies that would prevent it from assembling a minimal context for a test file with a single test function.

Key Assertions:

- The source code of the test file should contain a single test function.
- The ContextAssembler should be able to assemble a minimal context for the test file without any additional configuration.
- The assembled context should only contain the test function itself, without any other test functions or 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	30 lines (ranges: 33, 49, 52, 55, 58-59, 65, 78-79, 82-83, 86-87, 92, 94, 98-101, 103-112, 116)

PASSED tests/test_prompts.py::TestContextAssembler::test_balanced_context_limits 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the ContextAssembler to assemble a test case with balanced context limits.

Why Needed: This test prevents a potential bug where the assembler does not truncate long content within the context limit.

Key Assertions:

- The 'f1.py' file in the test result should be present in the assembled context.
- The 'f1.py' file should contain a message indicating truncation due to exceeding the context limit.
- The length of the 'f1.py' file within the context should not exceed 40 bytes (20 bytes + truncation message).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/prompts.py	34 lines (ranges: 33, 49, 52, 55, 58, 60-61, 65, 78-79, 82-84, 132, 135-138, 140-141, 144-145, 148, 151-152, 154-156, 158-159, 163, 191-192, 194)

PASSED `tests/test_prompts.py::TestContextAssembler::test_get_test_source_edge_cases` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Verify that the ContextAssembler correctly handles non-existent files and nested test names with parameters.

Why Needed: This test prevents a bug where the ContextAssembler returns an empty string for a file that does not exist, as well as a regression where the assembler fails to extract test source code from a nested test name with parameters.

Key Assertions:

- The function `_get_test_source` should return an empty string for a non-existent file.
- The function `_get_test_source` should correctly extract test source code from a nested test name with parameters.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 based on their glob patterns.

Why Needed: This test prevents a potential bug where the assembly process incorrectly excludes files that are not intended to be excluded.

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: 147)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`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: Consecutive lines in a list of numbers should be compressed to '1-3' using range notation.

Why Needed: This test prevents regression where consecutive lines are not compressed correctly.

Key Assertions:

- The function compress_ranges() should return the correct result for the input [1, 2, 3].
- The function compress_ranges() should handle cases with one or more consecutive numbers in the list.
- The function compress_ranges() should preserve the original order of numbers when there are no consecutive lines.
- The function compress_ranges() should not return an empty string for the input [1], [2, 3].
- The function compress_ranges() should handle cases with multiple consecutive lines in the list.
- The function compress_ranges() should preserve the original order of numbers when there are no consecutive lines.
- The function compress_ranges() should not return an empty string for the input [1, 2, 3, 4].

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`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: Verifies that the `compress_ranges` function correctly handles duplicate ranges.

Why Needed: Prevents a potential bug where the function incorrectly identifies non-duplicate ranges as duplicates.

Key Assertions:

- The input list has at least two elements with the same value.
- The first element of each range is within the valid range (1-3).
- Each unique range has a start value between 1 and 3, inclusive.
- No duplicate ranges are present in the input list.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

PASSED tests/test_ranges.py::TestCompressRanges::test_empty_list 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Testing the `compress_ranges` function with an empty input list.

Why Needed: This test prevents a potential bug where the function returns an incorrect result for an empty list.

Key Assertions:

- The function should return an empty string when given an empty list as input.
- The function should not raise any errors or exceptions when given an empty list as input.
- The function should correctly handle the case of an empty list without any additional computations.
- The function's output should be consistent with the expected result for an empty list (an empty string).
- Any side effects of the function should not affect its behavior in this specific test scenario.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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

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: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)

PASSED `tests/test_ranges.py::TestCompressRanges::test_non_consecutive_lines` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Non-consecutive lines in the input list are expected to be comma-separated.

Why Needed: This test prevents regression when non-consecutive lines are present in the input data.

Key Assertions:

- The function compresses the input list into a string with commas separating consecutive integers.
- The function handles cases where non-consecutive lines are present in the input list.
- Non-consecutive lines are correctly identified and separated by commas.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 require range notation.

Why Needed: This test prevents a potential bug where the function incorrectly assumes a single-element list and uses range notation.

Key Assertions:

- assert compress_ranges([5]) == '5'
- assert compress_ranges([]) == ''
- assert compress_ranges([1, 2, 3]) == [1, 2, 3]
- assert compress_ranges([-1, -2, -3]) == [-1, -2, -3]
- assert compress_ranges([5, 10, 15]) == [5, 10, 15]
- assert compress_ranges([[1, 2], [3, 4]]) == [[1, 2], [3, 4]]
- assert compress_ranges([]) is None

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`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 'test_two_consecutive' test verifies that two consecutive lines in a list of integers are compressed to the format 'start-end'.

Why Needed: This test prevents regression where non-consecutive lines are incorrectly compressed.

Key Assertions:

- assert compress_ranges([1, 2]) == '1-2'
- assert compress_ranges([3, 4]) == '3-4'
- assert compress_ranges([]) == ''
- assert compress_ranges([-1, -2]) == '-1-2'
- assert compress_ranges([10, 20]) == '10-20'
- assert compress_ranges([5, 6]) == '5-6'
- assert compress_ranges([7, 8]) == '7-8'

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	12 lines (ranges: 29, 33, 35-37, 39-40, 42, 50, 52, 65, 67)

PASSED `tests/test_ranges.py::TestCompressRanges::test_unsorted_input` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test 'test_unsorted_input' verifies that the function handles unsorted input correctly.

Why Needed: This test prevents a potential bug where the function would incorrectly group ranges in an unsorted input.

Key Assertions:

- The input list is sorted in ascending order.
- The output string contains only one range, as expected.
- The function groups each number individually within its assigned range.
- The function does not attempt to merge overlapping ranges.
- The function handles empty input lists correctly.
- The function raises an error when given unsorted input.
- The function maintains the original order of numbers in the input list.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`src/pytest_llm_report/util/ranges.py`	`16 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67)`

PASSED `tests/test_ranges.py::TestExpandRanges::test_empty_string` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Testing the `expand_ranges` function with an empty string.

Why Needed: This test prevents a potential bug where an empty string is not handled correctly by the `expand_ranges` function.

Key Assertions:

- The `expand_ranges` function should return an empty list when given an empty string as input.
- The `expand_ranges` function should handle empty strings without raising any exceptions or producing unexpected results.
- The `expand_ranges` function should produce a correct result for the input '...' (empty string).

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`src/pytest_llm_report/util/ranges.py`	`2 lines (ranges: 81-82)`

PASSED `tests/test_ranges.py::TestExpandRanges::test_mixed` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: test_mixed verifies that the `expand_ranges` function correctly handles mixed ranges and singles.

Why Needed: This test prevents a potential bug where the function would incorrectly handle ranges with overlapping values.

Key Assertions:

- The function should return an empty list when given a range with no valid values.
- The function should return a list containing all single values in the input string.
- The function should correctly handle ranges with overlapping values by returning a list of all valid values.
- The function should not throw any errors for invalid input, such as a range with negative numbers or non-numeric characters.
- The function should preserve the original order of the input values.
- The function should return an empty list when given a single value.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`src/pytest_llm_report/util/ranges.py`	`11 lines (ranges: 81, 84-91, 93, 95)`

PASSED `tests/test_ranges.py::TestExpandRanges::test_range` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The 'expand_ranges' function is expected to correctly handle a range input.

Why Needed: This test prevents the function from expanding the range incorrectly, potentially leading to incorrect output or errors.

Key Assertions:

- Input: '1-3'
- Expected Output: [1, 2, 3]
- The function should correctly split the range into individual numbers
- The function should handle negative numbers and zero correctly
- The function should not expand the range beyond its original bounds
- The function should raise an error for invalid input (e.g. 'a-b')
- The function should preserve order of elements in the range

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`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 are inverses.

Why Needed: This test prevents a potential bug where the order of ranges is not preserved when expanding them back to their original form.

Key Assertions:

- compressed should be equal to original
- expanded should be equal to original

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	27 lines (ranges: 29, 33, 35-37, 39-40, 42, 45-47, 50, 52, 65-67, 81, 84-91, 93, 95)

PASSED `tests/test_ranges.py::TestExpandRanges::test_single_number` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that the `expand_ranges` function correctly handles a single number input.

Why Needed: This test prevents potential bugs where the function incorrectly expands ranges for single numbers.

Key Assertions:

- The input string should be a single number (e.g., '5')
- The output list should contain only one element (i.e., [5])

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/ranges.py	7 lines (ranges: 81, 84-87, 93, 95)

📄 tests/test_render.py

9 tests

PASSED `tests/test_render.py::TestFormatDuration::test_milliseconds` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the `format_duration` function correctly formats durations in milliseconds for times less than 1 second.

Why Needed: This test prevents a regression where the duration is incorrectly formatted as seconds when it's less than 1 second.

Key Assertions:

- The function should return '500ms' for a duration of 0.5 seconds.
- The function should return '1ms' for a duration of 0.001 seconds.
- The function should return '0ms' for a duration of 0.0 seconds.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	2 lines (ranges: 65, 67)

PASSED `tests/test_render.py::TestFormatDuration::test_seconds` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the function formats duration values in seconds for inputs greater than or equal to 1 second.

Why Needed: This test prevents a potential regression where the function does not correctly format durations for input values greater than 1 second.

Key Assertions:

- The function should return '1.23s' when given an input of 1.23 seconds.
- The function should return '60.00s' when given an input of 60 seconds.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`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: Test that all outcomes map to CSS classes.

Why Needed: Prevents regression of CSS class mapping when different outcomes are encountered.

Key Assertions:

- The function outcome_to_css_class() correctly maps each outcome to a corresponding CSS class.
- The function outcome_to_css_class() handles the special case of 'xfailed' by returning 'outcome-xfailed'.
- The function outcome_to_css_class() correctly maps the string 'error' to 'outcome-error'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`src/pytest_llm_report/render.py`	`8 lines (ranges: 79-85, 87)`

PASSED `tests/test_render.py::TestOutcomeToCssClass::test_unknown_outcome` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the 'outcome_to_css_class' function with an unknown outcome.

Why Needed: Prevents a potential bug where the function returns incorrect CSS classes for unknown outcomes.

Key Assertions:

- The function should return 'outcome-unknown' when given an unknown outcome.
- The function should not return any other class than 'outcome-unknown'.
- The function should handle cases where the outcome is not a string.
- The function should raise an error if it encounters an unknown outcome.
- The function should log or report the unknown outcome in some way.
- The function should not silently fail without logging or reporting the issue.
- The function should provide a clear and descriptive message when returning 'outcome-unknown'.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`src/pytest_llm_report/render.py`	`8 lines (ranges: 79-85, 87)`

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_basic_report` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: The test verifies that a complete HTML document is rendered for a basic report.

Why Needed: This test prevents regression where the report rendering fails due to missing or incomplete plugin and repository information.

Key Assertions:

- The presence of '' in the rendered HTML.
- The presence of 'Test Report' in the rendered HTML.
- The presence of 'test::passed' in the rendered HTML.
- The presence of 'test::failed' in the rendered HTML.
- The presence of 'PASSED' and 'FAILED' in the rendered HTML.
- The presence of 'Plugin: v0.1.0' in the rendered HTML.
- The presence of 'Repo: v1.2.3' in the rendered HTML.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	52 lines (ranges: 65-67, 79-85, 87, 121-124, 126-127, 131-132, 141-143, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED `tests/test_render.py::TestRenderFallbackHtml::test_renders_coverage` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test renders coverage to ensure that the test suite includes necessary information.

Why Needed: This test prevents a regression where the coverage report is missing or incomplete.

Key Assertions:

- The test verifies that the `render_fallback_html` function returns HTML containing the source code of `src/foo.py` with 5 lines.
- The test checks that the generated HTML includes the specified number of lines (5) from the `src/foo.py` 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/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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.

Key Assertions:

- LlmAnnotation scenario: Tests login flow
- Prevents auth bypass why_needed: Prevents security vulnerability

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	54 lines (ranges: 65, 67, 79-85, 87, 121-124, 126-127, 131-134, 136-137, 141-142, 145-153, 158-160, 196, 229-236, 239-245, 248-249)

PASSED tests/test_render.py::TestRenderFallbackHtml::test_renders_source_coverage 1ms 🛡 3

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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: Test renders xpass summary for a report with two test cases.

Why Needed: This test prevents rendering of the summary section when there are no test failures or passes.

Key Assertions:

- The 'XFailed' and 'XPassed' tags should be present in the rendered HTML.
- Both 'XFailed' and 'XPassed' tags should contain the corresponding numbers (1 for xfailed, 1 for xpassed).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 Compute Sha256: Different content should produce different hashes.

Why Needed: This test prevents a potential bug where two different inputs to the `compute_sha256` function could produce the same output, potentially leading to incorrect reporting of different content.

Key Assertions:

- hash1 != hash2
- hash1 is not equal to hash2
- hash1 is not the same as hash2
- hash1 is different from hash2
- hash1 does not match hash2
- different hashes are produced for two different inputs

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 the same hash as another empty byte string.

Why Needed: This test prevents a potential bug where different inputs to `compute_sha256()` could produce different hashes due to differences in input data.

Key Assertions:

- Input 'b'' should have the same SHA-256 hash as Input 'b''
- Hash length of Input 'b'' should be 64 characters
- Empty byte string should return a hash with zero 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: 380-381, 384, 388-390)
`src/pytest_llm_report/report_writer.py`	1 lines (ranges: 55)

PASSED `tests/test_report_writer.py::TestReportWriter::test_build_run_meta` 5ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the ReportWriter to ensure it builds run metadata correctly.

Why Needed: This test prevents regression where the report writer does not include version info in the build run metadata.

Key Assertions:

- The duration of the test should be exactly 60 seconds.
- The pytest version should have a value.
- The plugin version should be '0.1.0'.
- The Python version should also match 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/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	67 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300)

PASSED `tests/test_report_writer.py::TestReportWriter::test_build_summary_all_outcomes` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test verifies that the `build_summary` method counts all outcome types correctly.

Why Needed: This test prevents a regression where the summary does not count all outcome types, potentially leading to incorrect reporting of test results.

Key Assertions:

- The total number of outcomes in the summary should be equal to the total number of tests.
- The number of passed outcomes in the summary should be equal to the number of passed tests.
- The number of failed outcomes in the summary should be equal to the number of failed tests.
- The number of skipped outcomes in the summary should be equal to the number of skipped tests.
- The number of xfailed outcomes in the summary should be equal to the number of xfailed tests.
- The number of xpassed outcomes in the summary should be equal to the number of xpassed tests.
- The number of error outcomes in the summary should be equal to the number of error tests.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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: When building a summary with multiple tests, the total count of passed and failed tests should be accurate.

Why Needed: This test prevents regression in cases where there are many tests but only one or two have been marked as 'failed'.

Key Assertions:

- The total number of passed tests is equal to the sum of their outcomes.
- The total number of failed tests is equal to the number of tests with an outcome of 'failed'.
- The total number of skipped tests is zero.
- The count of passed tests should be less than or equal to the count of failed tests.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 class does not initialize correctly with the given config.

Key Assertions:

- The `config` attribute of the `ReportWriter` instance should be equal to the provided `Config` object.
- The `warnings` list of the `ReportWriter` instance should be empty.
- The `artifacts` list of the `ReportWriter` instance should be empty.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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.

Why Needed: This test prevents regression where the report does not include all tests, which could be misleading for users.

Key Assertions:

- - len(report.tests) == 2
- - report.summary.total == 2

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	93 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-320, 330)

PASSED `tests/test_report_writer.py::TestReportWriter::test_write_report_includes_coverage_percent` 6ms 🛡️ 4

AI ASSESSMENT

Scenario: The test verifies that the `ReportWriter` class writes a report with an accurate total coverage percentage.

Why Needed: This test prevents a potential regression where the report does not accurately reflect the total coverage percentage, potentially leading to incorrect reporting or analysis.

Key Assertions:

- The `coverage_total_percent` attribute of the report summary is set to the provided value.
- The `report.write_report()` method returns an instance with the specified `summary` attribute.
- The `report.summary.coverage_total_percent` attribute is accessed and compared to the expected value in the assertion.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 regression where the report does not include source coverage information, potentially misleading users about the code's quality and maintainability.

Key Assertions:

- The length of the `source_coverage` list in the report should be 1.
- The file path of the first element in `source_coverage` should match 'src/foo.py'.
- The coverage percentage of the first element in `source_coverage` should be 87.5%.
- The covered lines of the first element in `source_coverage` should be 7 out of 8.
- The missed lines of the first element in `source_coverage` should be 1 out of 8.
- The coverage percentage of the first element in `source_coverage` should be greater than or equal to 87.5%.
- The covered ranges of the first element in `source_coverage` should match '1-4, 6-7'.
- The missed ranges of the first element in `source_coverage` should not match '5'.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`src/pytest_llm_report/report_writer.py`	92 lines (ranges: 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202-206, 211-218, 222, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314, 330)

PASSED tests/test_report_writer.py::TestReportWriter::test_write_report_merges_coverage 5ms 🛡️ 4

AI ASSESSMENT

Scenario: Test ReportWriter::test_write_report_merges_coverage verifies that the test writes a merged coverage report.

Why Needed: This test prevents regression by ensuring that the test writer correctly merges coverage into tests.

Key Assertions:

- The number of files covered in the report should be equal to the number of tests.
- Each file path in the coverage data should match the file path of a test.
- All lines in each file range should be present in the coverage data.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 ReportWriterWithFiles test falls back to direct write if atomic write fails and a cross-device link is attempted.

Why Needed: This test prevents a regression where the file system is unable to handle an atomic write operation, potentially leading to data loss or corruption.

Key Assertions:

- The report.json file should exist at the specified path.
- Any warnings generated by the ReportWriterWithFiles test should have code 'W203'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	67 lines (ranges: 235-237, 239, 241, 370-386, 388-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516-518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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: The test verifies that the `ReportWriter` creates an output directory if it doesn't exist.

Why Needed: This test prevents a bug where the report writer fails to create the output directory when it is missing.

Key Assertions:

- The output directory should be created with the correct name (`subdir/report.json`).
- The `tmp_path` directory should have been created with the correct parent directory (`/subdir`) and file name (`report.json`).
- The `ReportWriter` instance should have successfully written to the report file.
- The test result for `test1` should be marked as passed if the output directory is correctly created.
- The test result for `test2` should not be marked as passed if the output directory is incorrectly 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	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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)

src/pytest_llm_report/report_writer.py	123 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222-223, 226, 230, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 340, 343-345, 348-349, 352-354, 357, 360-364, 470-477, 495, 497, 499-501, 503, 506)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_ensure_dir_failure` 1ms 🛡️ 4

AI ASSESSMENT

Scenario: Test the `test_ensure_dir_failure` function to ensure it captures warnings when directory creation fails.

Why Needed: This test prevents a potential bug where the function does not report any warnings if directory creation is unsuccessful.

Key Assertions:

- The function should raise an error with a non-zero exit code (W201) for a directory that cannot be created.
- Any warning messages should have a code of 'W201'.
- The `writer.warnings` list should contain at least one warning object with a non-zero exit code.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/report_writer.py	12 lines (ranges: 156-158, 470-473, 480-484)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_git_info_failure` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the `get_git_info` function handles git command failures gracefully by returning `None` for both SHA and dirty flags.

Why Needed: This test prevents a potential bug where the `get_git_info` function fails to return any information when it encounters a git command failure, potentially leading to incorrect test results or errors.

Key Assertions:

- The `get_git_info` function should return `None` for both SHA and dirty flags if the git command fails.
- The `sha` variable should be set to `None` after calling `get_git_info()`.
- The `dirty` variable should also be set to `None` after calling `get_git_info()`.
- An exception of type `Exception` should be raised when calling `subprocess.check_output` with a side effect.
- The test should not pass if the git command fails, as it would return incorrect information about the repository.
- The test should only fail if the `git` executable is not found on the system.
- The test should handle the case where the git command returns an error message instead of raising an exception.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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` 35ms 🛡️ 6

AI ASSESSMENT

Scenario: Test 'test_write_html_creates_file' verifies that the report writer creates an HTML file.

Why Needed: This test prevents a regression where the report writer fails to create an HTML file when the configuration is set to write to a specific file path.

Key Assertions:

- The report writer should create an HTML file at the specified location.
- The report writer should contain the expected content in the HTML file.
- The HTML file should contain all test results with their respective outcomes and messages.
- The HTML file should not be empty.
- All required sections of the HTML file should be present (PASSED, FAILED, SKIPPED, XFailed, XPassed, Errors).
- The report writer should write to a specific file path 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/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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` 35ms 🛡️ 6

AI ASSESSMENT

Scenario: Should include xfail outcomes in the HTML summary.

Why Needed: This test prevents regression that may occur when xfail summaries are not included in the report.

Key Assertions:

- The 'XFAILED' and 'XPASSED' tags should be present in the HTML summary.
- The text content of these tags should match the expected outcomes.
- The presence of any other tags or elements should not affect the validity of this test.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 the `ReportWriter` creates a JSON file with an associated hash.

Why Needed: This test prevents regression where the report writer fails to create a JSON file even when there are tests in the report.

Key Assertions:

- The `report.json` file should be created at the specified path.
- At least one artifact should be tracked for the report.
- The number of artifacts should be greater than zero.
- The `ReportWriter` should write a JSON file to the specified location.
- The `ReportWriter` should create an associated hash for the report.
- The file created by the `ReportWriter` should exist at the specified path.
- The number of tests in the report should be tracked as an artifact.

COVERAGE

src/pytest_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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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` 37ms 🛡️ 6

AI ASSESSMENT

Scenario: Test that `write_pdf` creates a PDF file when Playwright is available.

Why Needed: This test prevents regression where the report writer fails to create a PDF file even if Playwright is installed and available.

Key Assertions:

- The `report.pdf` path should be created.
- Any artifacts with the same path as the created PDF file should exist.
- The `write_pdf` function should write the PDF file contents to the specified path.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	2 lines (ranges: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/render.py	25 lines (ranges: 30-31, 40, 42-46, 50-51, 53, 65, 67, 79-85, 87, 99, 101-102, 107)
src/pytest_llm_report/report_writer.py	125 lines (ranges: 55, 67-74, 76-81, 83-84, 98-99, 102, 105-108, 110, 127-128, 130, 156-158, 186, 192-193, 197-198, 202, 211-218, 222, 226, 230-231, 233, 254, 256-259, 262-264, 266, 268-275, 277-278, 280-289, 291-294, 296-297, 299-300, 312, 314-315, 317-318, 330, 401, 410, 412, 414-423, 434-435, 437-443, 448, 453, 455, 458-462, 470-471)

PASSED `tests/test_report_writer.py::TestReportWriterWithFiles::test_write_pdf_missing_playwright_warns` 5ms 🛡️ 4

AI ASSESSMENT

Scenario: Test that a warning is raised when PDF output is requested without Playwright.

Why Needed: To prevent a potential issue where the test fails due to missing Playwright for PDF output.

Key Assertions:

- The file 'report.pdf' should not exist.
- Any warnings with code WarningCode.W204_PDF_PLAYWRIGHT_MISSING should be present.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/options.py`	2 lines (ranges: 107, 147)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`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

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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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: Test verifies that report_writer_metadata_skips function skips metadata when reports are disabled.

Why Needed: This test prevents a bug where the report writer does not include metadata in the report even though the reports are enabled.

Key Assertions:

- The 'start_time' key should be present in the run meta.
- Metadata should contain the 'llm_model' key with default value None.
- Metadata should not contain the 'start_time' key when 'llm_model' is None.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/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: 107, 147)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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` correctly creates a schema from a dictionary.

Why Needed: Prevent regression in handling malformed input data where some required fields are missing.

Key Assertions:

- assert the correct field names are extracted from the dictionary
- assert all required keys are present in the schema
- assert the correct confidence level is applied to the assertion results

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/llm/schemas.py`	5 lines (ranges: 77-81)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)

PASSED `tests/test_schemas.py::TestAnnotationSchema::test_to_dict_full` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test that the `to_dict` method returns a dictionary with all required fields and their expected values.

Why Needed: Prevents regressions where the schema is not correctly converted to a dictionary, potentially leading to incorrect data being stored or retrieved.

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: 380-381, 384, 388-390)`

📄 tests/test_smoke_pytester.py

15 tests

PASSED `tests/test_smoke_pytester.py::TestBasicReportGeneration::test_html_report_created` 86ms 🛡️ 8

AI ASSESSMENT

Scenario: Test that an HTML report is created when the test function `test_simple` is run.

Why Needed: This test prevents a regression where the report generation might not be working correctly if the test function is not executed.

Key Assertions:

- The file 'report.html' should exist at the expected path.
- The content of the 'report.html' file should contain '' and 'test_simple' keywords.
- The 'test_simple' keyword should be present in the HTML report.

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	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)

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` 123ms 🛡️ 8

AI ASSESSMENT

Scenario: test_html_summary_counts_all_statuses verifies that the HTML summary counts include all statuses.

Why Needed: This test prevents regression where the HTML summary counts do not include all statuses, such as when a test fails or is skipped.

Key Assertions:

- asserts that each label in `labels` contains an integer value equal to its corresponding count from the report.
- asserts that any missing labels are reported with a message indicating their corresponding count.
- verifies that the 'Total Tests' label has a count of 6, which is correct based on the number of tests run.
- verifies that each status (Passed, Failed, Skipped) has a count equal to its actual occurrence in the report.
- verifies that any missing statuses are reported with a message indicating their corresponding count.
- checks that the 'Errors' label does not have a count of 0, which is correct based on the number of tests run.
- checks that all labels are present and contain an integer value equal to its corresponding count from the report.

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`	`46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)`

src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
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` 72ms 🛡️ 7

AI ASSESSMENT

Scenario: The JSON report is created and contains the expected schema version, summary statistics, and number of tests.

Why Needed: This test prevents a regression where the JSON report is missing or incorrectly formatted due to changes in the pytester library.

Key Assertions:

- The `schema_version` key should be present in the report data.
- The `summary` key should contain an object with keys `total`, `passed`, and `failed`.
- The `summary` object should have a non-negative value for `total`.
- The number of passed tests (`data['summary']['passed']`) should be equal to the total number of tests (2 in this case).
- The number of failed tests (`data['summary']['failed']`) should be equal to the total number of tests (2 in this case).

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	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71,

	73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
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` 77ms 🛡️ 13

AI ASSESSMENT

Scenario: Verify that LLM annotations are included in the report for a provider with llm_report_provider=litellm and llm_report_model=gpt-4o-mini.

Why Needed: Prevents regressions by ensuring LLM annotations are present in the report, which is critical for maintaining test coverage and reliability.

Key Assertions:

- The scenario 'Checks the happy path' is included in the LLM annotation.
- The reason 'Prevents regressions' is included in the LLM annotation.
- 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	23 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 66-67, 69, 78, 99-100, 102)
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	47 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	186 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-333, 336, 338, 341-345, 348, 350-355, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
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 86ms 🛡️ 13

AI ASSESSMENT

Scenario: Test that LLM errors are surfaced in HTML output.

Why Needed: Prevents regression where LLM errors are not reported correctly.

Key Assertions:

- The test verifies that the 'LLM error' is present in the report.
- The test verifies that the string 'boom' is present in the report.
- The test checks if the content of the report contains both 'LLM error' and 'boom'.

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	21 lines (ranges: 37-38, 44, 46, 49, 51-52, 54-60, 62-63, 80, 82, 99-100, 102)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	47 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-259, 261, 263-265, 270-272,

	274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	186 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203-205, 207-208, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-333, 336, 338, 341-346, 350-355, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
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 57ms 🛡️ 7

AI ASSESSMENT

Scenario: Test the LLM opt-out marker to ensure it records the correct marker.

Why Needed: This test prevents regression where the LLM opt-out marker is not recorded correctly.

Key Assertions:

- The 'llm_opt_out' marker should be present in the report.json file.
- The 'llm_opt_out' marker should be set to True for all tests.
- The 'llm_opt_out' marker should only be present for one test.
- The number of tests with 'llm_opt_out' marker should be 1.
- The 'llm_opt_out' marker should not be False for any 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, 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	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175,

	177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
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 56ms 🛡️ 7

AI ASSESSMENT

Scenario: Test the requirement marker functionality.

Why Needed: This test prevents a bug where the requirement marker is not recorded for tests with multiple requirements.

Key Assertions:

- The 'requirement' marker should be applied to each test function.
- Each test function should have at least one requirement marked.
- The 'requirements' key in the test data should contain both 'REQ-001' and 'REQ-002'.
- The 'reqs' list in the test data should contain both 'REQ-001' and 'REQ-002'.

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-200, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	76 lines (ranges: 162, 166-171, 173, 175, 177, 179, 182, 184, 186, 188, 190, 192, 194-196, 370-386, 388, 391, 393, 396-399, 401, 403, 405, 407, 409, 413, 425, 455-463, 465, 467, 506, 508-512, 514, 516, 518, 520, 522, 524, 526, 528)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213,

	238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
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` 63ms 🛡️ 7

AI ASSESSMENT

Scenario: Test 'test_multiple_xfail_outcomes' verifies that multiple xfailed tests are recorded in the report.

Why Needed: This test prevents regression and ensures that the correct number of xfailed tests is reported when running pytest with the '--llm-report-json' option.

Key Assertions:

- The 'summary' key in the report JSON contains the correct count of xfailed tests.
- Each xfailed test has a corresponding outcome ('xfailed') in the list of outcomes.
- The test outcomes are correctly matched with their respective test names.

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	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175,

	177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
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` 56ms 🛡️ 7

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	43 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 106-107, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163, 167-169, 171, 181, 185-186, 198-199, 209-210, 250-251, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	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	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)

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` 59ms 🛡️ 7

AI ASSESSMENT

Scenario: Verifies that an xfailed test is recorded in the report.

Why Needed: Prevents regression by ensuring that xfailed tests are properly reported.

Key Assertions:

- The number of xfailed tests should be exactly 1.
- The value of `xfailed` in the JSON report should be 1.
- The test function `test_xfail()` should fail.

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	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269,

	276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
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` 60ms 🛡️ 7

AI ASSESSMENT

Scenario: Test parameterized tests are recorded separately.

Why Needed: This test prevents regression in the parametrization of tests, ensuring that all tests are run with the correct parameters.

Key Assertions:

- The total number of tests should be 3 (1 passed, 2 failed).
- All tests should have been run with the correct parameters (x = [1, 2, 3]).

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	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272-274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284,

	287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
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` 50ms 🛡️ 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	45 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	118 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 380-381, 384, 388-390)

PASSED `tests/test_smoke_pytester.py::TestPluginRegistration::test_markers_registered` 45ms 🛡️ 3

AI ASSESSMENT

Scenario: Verify that LLM markers are registered in pytester.

Why Needed: Prevents a potential bug where LLM markers are not registered, potentially causing issues with test execution.

Key Assertions:

- The `pytester` instance is run with the `--markers` flag.
- Llama model output lines (`*llm_opt_out*`, `*llm_context*`) match expected patterns.
- Requirement marker line (`*requirement*`) matches expected 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/options.py	45 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	118 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 380-381, 384, 388-390)

PASSED tests/test_smoke_pytester.py::TestPluginRegistration::test_plugin_registered 52ms 🛡️ 3

AI ASSESSMENT

Scenario: The plugin is successfully registered by pytest11 and the help message is displayed.

Why Needed: This test prevents a potential issue where the plugin registration fails or is not visible in the help message.

Key Assertions:

- The `--help` option is available when running pytest11.
- The output of the `--help` option contains the string 'llm-report'.
- The `pytester.runpytest()` method returns a result object that includes an `stdout` attribute with a matching line.
- The `stdout.fnmatch_lines()` method matches the expected lines ('*--llm-report*').

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/options.py	45 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270, 272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	118 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 380-381, 384, 388-390)

PASSED tests/test_smoke_pytester.py::TestSpecialCharacters::test_special_chars_in_nodeid 89ms 🛡️ 8

AI ASSESSMENT

LLM error: Failed to parse LLM response as JSON

COVERAGE

src/pytest_llm_report/collector.py	40 lines (ranges: 78-79, 90, 93-94, 96, 99-100, 104, 109-112, 114-115, 124, 127, 132-133, 140, 155-159, 163-164, 167-169, 171, 181, 185-186, 198-199, 209-210, 277, 285)
src/pytest_llm_report/coverage_map.py	12 lines (ranges: 44-45, 58-60, 72-73, 83, 86, 88-90)
src/pytest_llm_report/errors.py	4 lines (ranges: 139-142)
src/pytest_llm_report/models.py	1 lines (ranges: 162)
src/pytest_llm_report/options.py	46 lines (ranges: 107, 147, 175, 178-179, 185-186, 193-194, 201-202, 209, 211, 213, 215, 217, 220, 224, 248, 251-253, 255-259, 261, 263-265, 270-272, 274, 276, 278, 280, 282, 286, 288, 290, 292, 294, 298, 300)
src/pytest_llm_report/plugin.py	166 lines (ranges: 40, 43-47, 49-53, 55-59, 61-65, 67-71, 73-78, 80-85, 89-93, 95-99, 101-105, 107-111, 113-117, 121-124, 126-129, 131-134, 136-140, 142-145, 147-151, 153-156, 169-171, 173-175, 177-179, 183, 187-188, 190, 192, 195-196, 203, 212-213, 238, 242, 246, 249, 268-269, 276-277, 280-281, 283-284, 287-291, 293, 296-297, 299, 302-303, 324, 330-331, 358-368, 380-381, 384, 388-390, 401, 405, 424, 428-430, 441, 445, 448, 450-451)
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: Prevents a potential bug where the function does not correctly handle durations less than 60 seconds.

Key Assertions:

- The result should be '1m 0.0s' when the input is 60.0.
- The result should contain exactly one unit of time ('m') and zero seconds.
- The result should have a total duration of 60 seconds (i.e., '1m 00.0s').
- Any additional units of time should be omitted (e.g., '1h 30.0s' is not expected).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	6 lines (ranges: 39, 41, 43, 46-48)

PASSED `tests/test_time.py::TestFormatDuration::test_microseconds_format` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 0.0005 seconds.

Why Needed: Prevents a potential bug where the function incorrectly returns 'ms' instead of 'μs' for durations in microseconds.

Key Assertions:

- The result contains the string 'μs' to indicate the unit of measurement (microseconds).
- The duration is correctly converted from seconds to microseconds.
- The function handles negative durations without errors.
- The function does not return 'ms' for durations in microseconds.
- The function raises an AssertionError with a clear message when the input duration is 0.
- The function can handle very small positive and negative durations without issues.
- The function correctly handles cases where the input duration is zero.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`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 durations with milliseconds, potentially leading to incorrect results or unexpected behavior in downstream applications.

Key Assertions:

- The result of calling `format_duration(0.5)` contains the string 'ms' (milliseconds).
- The value returned by `format_duration(0.5)` is equal to '500.0ms'.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`src/pytest_llm_report/util/time.py`	3 lines (ranges: 39, 41-42)

PASSED `tests/test_time.py::TestFormatDuration::test_minutes_format` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the 'minutes' format for durations over a minute.

Why Needed: This test prevents regressions where the 'minutes' format is not correctly applied to durations over a minute.

Key Assertions:

- The result should contain the string 'm' (for minutes) and be in the format '1m 30.5s'.
- The duration value should be converted to seconds first, then appended with '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/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`src/pytest_llm_report/util/time.py`	6 lines (ranges: 39, 41, 43, 46-48)

PASSED `tests/test_time.py::TestFormatDuration::test_multiple_minutes` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of 3 minutes and 5 seconds.

Why Needed: This test prevents regression in the format_duration function when given durations with multiple minutes.

Key Assertions:

- The result should be '3m 5.0s' as expected.
- The total number of seconds should be calculated correctly.
- The minutes and seconds parts should be separated correctly.
- Any leading zeros in the seconds part should be preserved.
- The function should handle durations with multiple minutes correctly.
- The function should return a string that is human-readable and easy to understand.

COVERAGE

`src/pytest_llm_report/collector.py`	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
`src/pytest_llm_report/plugin.py`	6 lines (ranges: 380-381, 384, 388-390)
`src/pytest_llm_report/util/time.py`	6 lines (ranges: 39, 41, 43, 46-48)

PASSED tests/test_time.py::TestFormatDuration::test_one_second 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a duration of exactly one second.

Why Needed: Prevents a potential bug where the function incorrectly formats durations longer than one second as '1.00s'.

Key Assertions:

- The function should return '1.00s' when given a duration of exactly one second.
- The function should not attempt to format durations longer than one second.
- The function should handle negative durations correctly.
- The function should not silently ignore invalid input (e.g., non-numeric values).
- The function should raise an error for very large durations (e.g., > 10 seconds).
- The function should preserve the original unit of the input duration (e.g., milliseconds, microseconds).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED `tests/test_time.py::TestFormatDuration::test_seconds_format` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function to ensure it correctly formats seconds under a minute.

Why Needed: This test prevents regression in the `format_duration` function, which may have been changed to incorrectly format seconds under a minute in previous versions of the library.

Key Assertions:

- The result should contain the string 's' (i.e., 'seconds')
- The result should be equal to '5.50s'
- The function should correctly handle cases where the input is an integer greater than 1

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	4 lines (ranges: 39, 41, 43-44)

PASSED tests/test_time.py::TestFormatDuration::test_small_milliseconds 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `format_duration` function with a small millisecond duration.

Why Needed: This test prevents a potential bug where the function does not correctly format durations in milliseconds.

Key Assertions:

- The output of the `format_duration` function should be '1.0ms' for a duration of 1 millisecond.
- The function should handle millisecond durations without truncation or rounding errors.
- Any negative durations should be handled correctly by the function.
- The function should support durations in both decimal and fractional formats (e.g., 1.5s, 0.75ms).
- The function should not return 'nan' for invalid input values (e.g., zero duration, non-numeric value).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
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 the correct formatting of very small durations in microseconds.

Why Needed: This test prevents a potential bug where the 'microseconds' unit is not correctly formatted for very small values.

Key Assertions:

- The function `format_duration(0.000001)` returns the string '1µs'.
- The value of `result` matches the expected format '1µs'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	2 lines (ranges: 39-40)

PASSED `tests/test_time.py::TestIsoFormat::test_formats_datetime_with_utc` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Test the ISO format of datetime objects with UTC timezone.

Why Needed: Prevents a potential bug where datetime objects with UTC timezone are not correctly formatted as ISO.

Key Assertions:

- The result should be in the format 'YYYY-MM-DDTHH:MM:SS+HH:MM:SS'.
- The time zone offset should be correctly represented (+00:00).
- The date and time components should be separated by a space.
- No leading zeros for minutes, hours, or seconds.
- The UTC timezone offset should be displayed as 'UTC' instead of 'GMT' or 'Z'.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

PASSED `tests/test_time.py::TestIsoFormat::test_formats_naive_datetime` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that naive datetime is correctly formatted without timezone.

Why Needed: Prevents a potential bug where the naive datetime format may not match the expected output due to timezone differences.

Key Assertions:

- The function `iso_format(dt)` returns the correct ISO-formatted string for the given naive datetime.
- The resulting string is in the ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ).
- No timezone information is present in the output.
- The function handles different date and time formats correctly.
- The function does not introduce any timezone-related errors or inconsistencies.

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`src/pytest_llm_report/util/time.py`	`1 lines (ranges: 27)`

PASSED tests/test_time.py::TestIsoFormat::test_formats_with_microseconds 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Tests the `iso_format` function with a datetime object that includes microseconds.

Why Needed: This test prevents a potential issue where the `iso_format` function does not correctly format datetime objects with microseconds.

Key Assertions:

- The result of `iso_format(dt)` should contain the string '123456'.
- The result of `iso_format(dt)` should include the substring '123456' in its output.
- The `datetime` object passed to `iso_format(dt)` should be a valid datetime object.
- The `tzinfo` parameter of the `datetime` object passed to `iso_format(dt)` is set to UTC.
- The `result` variable should contain the string '123456' after calling `iso_format(dt).'
- The `result` variable should include the substring '123456' in its output after calling `iso_format(dt).'
- A valid datetime object with microseconds should be passed to `iso_format(dt)`.
- A datetime object without microseconds should not be passed to `iso_format(dt)`.
- A datetime object with an invalid timezone should not be passed to `iso_format(dt)`.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	1 lines (ranges: 27)

PASSED `tests/test_time.py::TestUtcNow::test_has_utc_timezone` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `utc_now()` function returns a datetime object with an associated UTC timezone.

Why Needed: Prevents regression in tests where the `datetime` module's default timezone is not set to UTC.

Key Assertions:

- The returned `datetime` object has a valid timezone.
- The returned `datetime` object's timezone is equal to `UTC`.
- The returned `datetime` object does not have an associated timezone (i.e., it is naive).
- The returned `datetime` object's timezone is set to the system's default timezone.
- The returned `datetime` object's timezone is set to a custom timezone specified by the `pytz` library.

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)

PASSED `tests/test_time.py::TestUtcNow::test_is_current_time` 1ms 🛡️ 3

AI ASSESSMENT

Scenario: Verifies that the `utc_now()` function returns a current time within a specified tolerance.

Why Needed: This test prevents regression in cases where the system is not aware of the UTC timezone.

Key Assertions:

- The result of `utc_now()` should be equal to or greater than the before time and less than or equal to the after time.
- The difference between the before time and the result should be within a specified tolerance (e.g., 1 second).
- The system should correctly handle cases where the current time is not yet known (i.e., `datetime.now()` returns before UTC).

COVERAGE

`src/pytest_llm_report/collector.py`	`14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)`
`src/pytest_llm_report/plugin.py`	`6 lines (ranges: 380-381, 384, 388-390)`
`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 regression in the case where the system is not properly initialized or has an incorrect timezone.

Key Assertions:

- result is an instance of `datetime`.
- result has a valid time zone.
- result is not None.
- result is a datetime object (e.g., `datetime.datetime.now()`).
- result is not a naive datetime object (i.e., it does not have an implicit timezone).

COVERAGE

src/pytest_llm_report/collector.py	14 lines (ranges: 90, 93, 96, 99, 110-112, 114-115, 124, 127, 140, 209-210)
src/pytest_llm_report/plugin.py	6 lines (ranges: 380-381, 384, 388-390)
src/pytest_llm_report/util/time.py	1 lines (ranges: 15)