CSC490 Assignment 6

Team:
Son Nguyen (ID 1009656560), Kyle (ID 1007785229),
Daniel (ID 1008035378), Jinbo (ID 1004821419)
Project: ArXplorer - Semantic Search for arXiv

Part One

We automated test coverage reporting by adding a GitHub Actions workflow that runs on every push and pull request to the active branches. The workflow sets up Python 3.10, installs the test dependencies from requirements-ci.txt plus a CPU-only PyTorch wheel, then exports PYTHONPATH to include src before running ‘pytest --cov=src --cov-report=xml’. It generates a coverage.svg badge, writes a brief summary to the PR via the step summary, and uploads coverage.xml and coverage.svg as artifacts, with automatic badge commits on branch a6. The current coverage is 100 % (lines-covered="1003" out of lines-valid="1003" in coverage.xml), so reviewers can see both pass/fail status and coverage directly in the PR checks without running tests locally.

Part Two

We drove coverage breadth to 100 % across the entire src tree by expanding the test suites to cover every module. The encoders (dense and sparse) now have tests for adapter switching, normalization, warning paths, and error branches; data and config tests validate loaders, documents, and settings parsing; indexer and searcher tests exercise Milvus hybrid indexing/search flows, filter expressions, and metadata retrieval; reranker tests cover CrossEncoder, Jina, and Qwen paths, including auto device handling and empty/missing text cases; and the query rewriter suite stresses JSON parsing, prompt fallbacks, and filter extraction. Running pytest --cov=src --cov-report=xml confirms full line coverage (lines-covered="1003" of lines-valid="1003"), and the GitHub Actions workflow publishes the badge and artifacts so reviewers can see the breadth of coverage on every push/PR.

Part Three

Target module: src/retrieval/query_rewriting/llm_rewriter.py (LLMQueryRewriter).
This is the most complex component in the repo: it detects canonical queries, builds prompts with/without chat templates, parses LLM JSON output, rewrites queries, extracts filters, and provides multiple fallback paths. Because half the assignment is "we'll try to break it," this module is ideal for a deep test suite.

Selected Edge Cases & Tests
All tests live in tests/test_query_rewriting.py; each bullet lists the scenario and the specific test(s). Positive ("✓") vs. negative/fallback ("✗").

1. ✓ **Canonical JSON happy path** (test_llm_rewriter_parses_canonical_json)
   - Ensures we decode canonical RR output correctly, respect num_rewrites, and still run pattern-check fallback.
2. ✗ **Malformed JSON fallback** (test_llm_rewriter_falls_back_to_pattern_detection, test_rewrite_empty_generated_text_defaults_to_query)
   - Forces JSON decode errors and blank outputs to confirm we return safe rewrites + canonical detection fallback.
3. ✓ **Device auto-detection + legacy modes** (test_llm_rewriter_defaults_to_cpu, test_rewrite_causal_lm_without_canonical)
   - Verifies we pick CPU on machines without CUDA and still return clean rewrites when canonical detection is disabled.
4. ✓/✗ **Prompt building, chat-template vs. fallback** (test_build_prompt_without_chat_template, test_build_prompt_canonical_no_chat_template)
   - Checks both canonical and legacy prompt flows when tokenizer lacks apply_chat_template.
5. ✓/✗ **num_rewrites clamping & padding** (test_rewrite_uses_default_num_rewrites)
   - Confirms we clamp to [1,10], pad missing rewrites with originals.
6. ✓/✗ **Filter extraction success** (test_extract_filters_and_rewrite_success) + **fallbacks** (test_extract_filters_and_rewrite_fallback, test_extract_filters_invalid_structure, test_extract_filters_no_chat_template)
   - Covers correct JSON parsing, clamped confidence, invalid structures (strings instead of dicts), no chat-template fallback.
7. ✗ **Seq2Seq rewrite path** (test_seq2seq_mode_returns_plain_rewrites, test_extract_filters_seq2seq_mode)
   - Runs the T5-style branch to confirm we still output clean strings in absence of canonical detection.
8. ✓ **Filter-only helper** (test_build_filter_expr_full_filters, test_build_filter_expr_year_only, test_build_filter_expr_none_returns_none)
   - Ensures year/citation filters compose correctly and return None for empty input.
9. ✗ **Canonical pattern detector** (test_detect_canonical_fallback_false)
   - Negative case: queries without canonical keywords should not be misclassified.
10. ✓ **Filter success path from raw JSON** (test_extract_filters_without_chat_template, test_extract_filters_no_chat_template)
    - Validates the "multi query" prompt fallback still yields correct filter dict.

Each of these is referenced in the module comments (lines 110-130) so graders can cross-check quickly.

Why these tests?

- They cover every branch that can fail in production (JSON parsing, prompt fallback, invalid configurations).
- They demonstrate both "positive" flows (LLM returns good JSON, filters parse) and "negative" flows (missing templates, raw garbage output), per the spec.
- They make the assignment's "we'll try to find edge cases" challenge easier because every known edge case we found is now codified and linked.

Execution & Coverage

- Running python -m pytest --maxfail=1 --cov=src --cov-report=xml exercises all 20 tests; coverage.xml shows lines-valid="1003" vs lines-covered="1003" (100 %).
- Partial test runs (e.g., only tests/test_query_rewriting.py) will regenerate coverage.xml with lower coverage, so final submissions should always re-run the full command.

Part Four

https://drive.google.com/file/d/17oqg_Ilcd2cheSC6h4feQy7GP1KOZGjz/view?usp=sharing