

Applied LLM Systems

Homework 1: Code Plagiarism Detection with RAG Systems

Assignment Overview

Build and evaluate a code plagiarism detection system using different retrieval and generation approaches. You will implement four methods—pure embedding search, direct LLM analysis, standard RAG, and hybrid RAG—and compare their effectiveness on a dataset you collect yourself.

Phase 1: Data Collection and Dataset Construction (25 points)

Collect code from at least five GitHub repositories in a single programming language. Construct a labeled dataset of minimum 30 test cases containing positive and negative examples of plagiarism.

Positive examples: Take functions from your repos and apply realistic transformations like variable renaming, comment removal, minor refactoring, or structure reordering while preserving core logic.

Negative examples: Write original code for similar tasks with fundamentally different implementations or use code from unrelated problem domains.

Document your collection process, transformations applied, and dataset rationale.

Phase 2: System Implementation (40 points)

Implement four plagiarism detection systems:

Pure Embedding Search: Use embedding models to represent code in vector space. Retrieve top-k similar snippets based on cosine similarity and determine plagiarism using similarity thresholds.

Direct LLM Analysis: Provide all relevant repository code in the LLM context window and ask the model to determine plagiarism. This tests what LLMs achieve with full information access.

Standard RAG: Use embedding-based retrieval to find relevant code snippets, then provide them as context to an LLM for final plagiarism determination.

Hybrid RAG: Combine dense embeddings with BM25 lexical matching in the retrieval stage. Fuse results from both methods before passing top candidates to the LLM.

Handle the complete pipeline from indexing to prediction for each system. Consider chunking strategies (function-level vs file-level), prompt engineering, and retrieval hyperparameters.

Phase 3: Evaluation and Analysis (25 points)

Evaluate each system on your test dataset using precision, recall, F1 score, and accuracy. Analyze error types for each approach and identify patterns in failures.

Conduct ablation studies: vary the number of retrieved documents (k) in RAG systems, experiment with different fusion weights in hybrid RAG, and measure performance impacts.

Address the core question: what are the trade-offs between retrieval quality, computational cost, context usage, and accuracy? When would you recommend each approach?

Phase 4: Code Quality and Best Practices (10 points)

Your implementation must follow professional software engineering standards with clean, well-documented, modular code.

Deliverables

Submit a **single ZIP file** containing the following structure:

```
homework4_yourname.zip
├── requirements.txt           # Python dependencies
├── 01_indexing.ipynb         # Indexing and data preparation
├── notebook
│   ├── 02_interactive.ipynb  # Interactive testing notebook
│   └── 03_evaluation.ipynb    # Evaluation and visualization
├── notebook
│   ├── data/
│   │   ├── reference_corpus/  # Collected GitHub repositories
│   │   └── test_dataset.json  # Labeled test cases
│   ├── indexes/              # Pre-built indexes (embeddings, BM25,
│   │   └── etc.)
│   └── results/
│       └── comparison_chart.png # Main results comparison
```

Notebook Specifications

01_indexing.ipynb - Indexing and Data Preparation

- Data collection code and chunking strategies
- Building and saving all indexes for the reference corpus
- All data structures saved to disk for use by other notebooks

02_interactive.ipynb - Interactive Testing

- Four callable functions: `detect_embedding()`, `detect_llm()`, `detect_rag()`, `detect_hybrid_rag()`
- Each function takes a code snippet as input and returns plagiarism detection results
- Loads pre-built indexes from `01_indexing.ipynb`

03_evaluation.ipynb - Evaluation and Visualization

- Loads test dataset and pre-built indexes
- Evaluates all four methods automatically
- Generates and saves comparison chart as PNG file
- Includes ablation studies and performance analysis

Required Files

Generated Artifacts: Include the generated comparison chart (PNG format) and pre-built indexes in your submission. While these should be provided for immediate grading, your notebook code must be capable of reproducing them from scratch.

Requirements

Execution Requirements

- All notebooks must run completely from top to bottom using "Restart and Run All"
- No manual intervention, cell skipping, or error handling required during execution
- All file paths must be relative and work across different systems
- Dependencies clearly specified in `requirements.txt`

Security Requirements

- API keys must not appear in any notebook or code file
- Use environment variables (e.g., `os.getenv('OPENAI_API_KEY')`) or config files

Code Quality Requirements

- Modular, well-documented functions
 - Consistent naming conventions and code style
 - Clear comments explaining complex logic
 - Appropriate error handling and validation
-

Grading Breakdown

Component	Points	Description
Dataset Quality	15	Repository diversity, realistic positive examples with documented transformations, appropriate negative examples, balanced dataset size, clear documentation
Implementation	40	All four systems correctly implemented (10 pts each): proper embeddings, RAG architectures, hybrid retrieval, chunking strategies, prompt engineering
Evaluation & Analysis	35	Comprehensive metrics, meaningful ablation studies, thoughtful failure analysis, comparative insights, clear visualizations
Code Quality	10	Clean modular code, proper documentation, best practices, notebook organization, reproducibility
TOTAL	100	

Automatic Deductions

Violation	Deduction	Description
Non-runnable Notebooks	-30%	If any notebook fails to run from top to bottom without errors
Exposed API Keys	-10%	If API keys are hardcoded in notebooks or code files

Due Date: Two weeks from assignment date

Submission: Upload your ZIP file to LMS before 11:59 PM on the due date.