Progress Report: LLMs as Evaluators for Code Metadata

Capstone Project

November 3, 2025

1 Project Overview

This project investigates the use of Large Language Models (LLMs) to evaluate and judge the quality of source code metadata, specifically focusing on code comments and documentation. The research employs Retrieval Augmented Generation (RAG) methods to enable LLMs to assess comment quality by leveraging structured program analysis and semantic context from real-world codebases.

The current phase has successfully implemented a complete pipeline that extracts structured metadata from a large-scale C codebase (libpng), creates navigable program maps, and deploys an LLM-powered agent capable of intelligent code navigation and metadata evaluation tasks.

2 Implementation and Methodology

2.1 Static Code Analysis Pipeline

The project implements a comprehensive static analysis system (`build_map.py`) that processes the libpng repository (a mature, production-grade C library) to extract three critical types of program metadata:

1. **Code Map**: Complete inventory of function definitions, including signatures, file locations, and line numbers, extracted using `ctags`
2. **File Dependencies**: Include graph mapping header dependencies between files, extracted via `ripgrep`
3. **Call Graph**: Function call relationships showing which functions are invoked by each source file, approximated using pattern matching

The analysis pipeline leverages industry-standard tools:

- `universal-ctags` for function definition extraction
- `ripgrep` for efficient pattern-based code analysis
- `graphviz` for generating scalable visualization outputs

2.2 LLM Agent Architecture

The agent system (`main.py`) integrates Google's Gemini 2.5 Flash model via the LlamaIndex framework. The agent is equipped with two specialized tools that enable retrieval-augmented code analysis:

- `get_code_map_info`: Queries structured metadata (code map, dependencies, or call graph) from the generated dataset
- `read_source_file`: Retrieves complete source code content from any file in the repository

The agent uses a ReAct (Reasoning + Acting) architecture, enabling it to:

- Navigate the codebase systematically
- Combine structural metadata with source code context
- Provide grounded answers about code functionality and metadata quality

3 Results and Outputs

3.1 Dataset Generation

The static analysis pipeline successfully processed the entire libpng codebase, generating a comprehensive structured dataset:

Metric	**Count**
Functions Indexed	2,548
Files with Dependencies	81
Files with Call Information	88

Table 1: Generated Dataset Statistics

The dataset is stored in `code_structure.json`, a structured JSON file containing all extracted metadata. This serves as the indexed input for LLM evaluation tasks.

3.2 Sample Dataset Structure

The generated `code_structure.json` contains three main sections. Below is an example entry from the code map:

Listing 1: Sample Function Entry from code_map

```
"png_get_cHRM": {
    "file": "pngget.c",
    "line": 526,
    "signature": "png_get_cHRM(png_const_structrp png_ptr,
                 png_const_inforp info_ptr,
                 double *whitex, double *whitey,
                 double *redx, double *redy,
                 double *greenx, double *greeny,
                 double *bluex, double *bluey)"
}
```

3.3 Visualization Outputs

The pipeline generates scalable SVG visualizations that aid in understanding code structure. These visualizations are automatically converted to PDF format for inclusion in documentation and analysis.

These visualizations provide human-readable representations of complex code relationships, enabling researchers to identify structural patterns and potential metadata evaluation targets. The focused views (Figures 2 and 3) are particularly useful for understanding local call patterns, while the dependency graph (Figure 1) reveals broader architectural relationships.

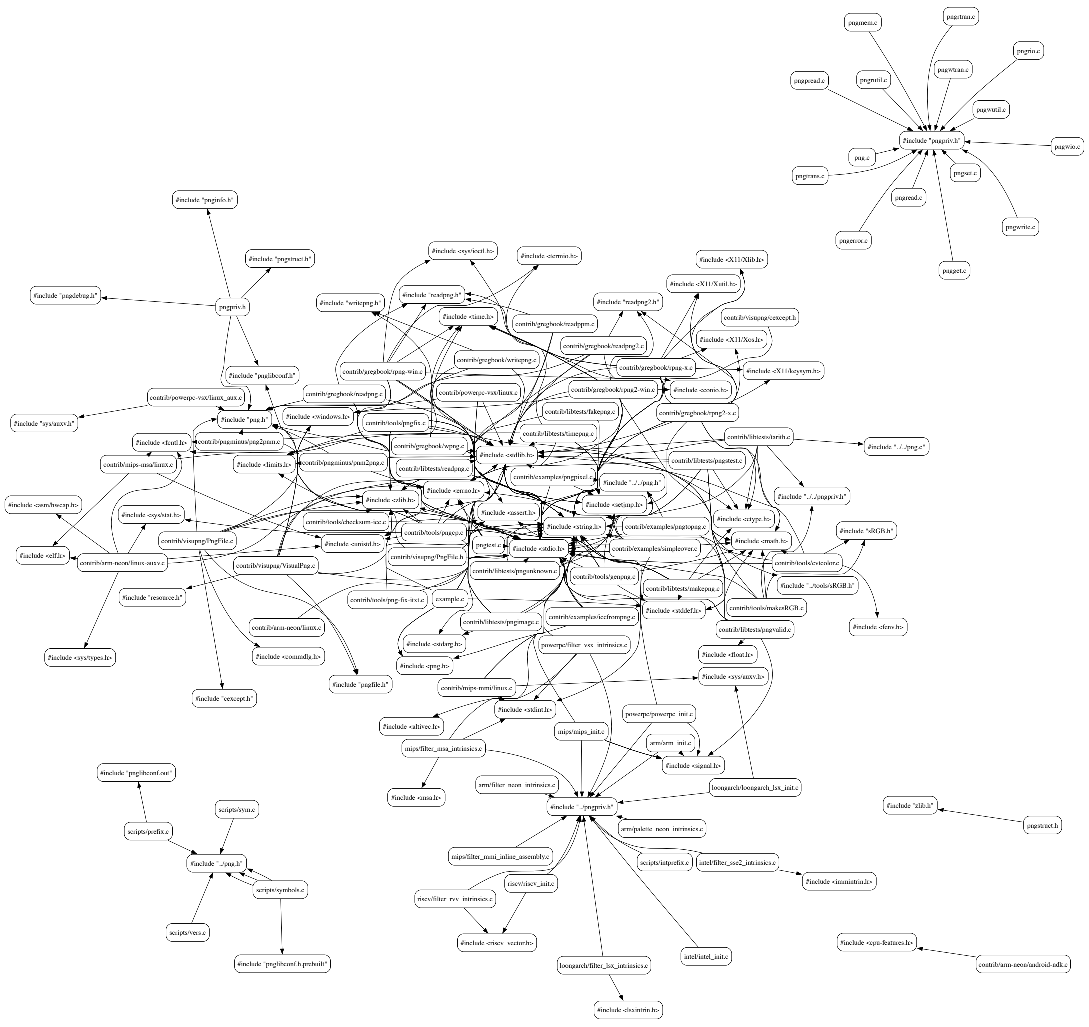

Figure 1: File Dependency Graph: Complete dependency visualization showing header inclusion relationships between source files in the libpng codebase. This graph helps identify module coupling and dependency hierarchies.

3.4 Agent Performance Demonstration

The LLM agent successfully demonstrates its capability to navigate the codebase and retrieve accurate information. In a test query asking for the function signature of `png_get_cHRM`, the agent:

1. Used the `get_code_map_info` tool to query structured metadata
2. Navigated through multiple tool calls to locate the relevant information
3. Retrieved the complete function signature with correct parameter types

Agent Response:

```
png_get_cHRM(png_const_structrp png_ptr, png_const_inforp info_ptr,
             double *whitex, double *whitey, double *redx, double *redy,
             double *greenx, double *greeny, double *bluex, double *bluey)
```

The agent's reasoning process is logged in `agent_log_*.txt` files, providing full transparency into the retrieval and reasoning steps.

Figure 2: Call Map for `pngread.c`: Focused visualization showing all functions called by the `pngread.c` module. This targeted view demonstrates the complexity of function call relationships within a single critical file.

Figure 3: Call Map for `contrib/visupng` Module: Call graph visualization for the `contrib/visupng` directory, showing function call relationships within this module and connections to other parts of the codebase.

4 Achievements

4.1 Technical Accomplishments

- **Complete Pipeline Implementation**: Successfully built an end-to-end system from raw source code to structured metadata and LLM-powered analysis
- **Scalable Architecture**: The pipeline can process large codebases (2,500+ functions) efficiently using parallel tool execution
- **Structured Dataset Generation**: Created a reusable JSON schema that can be extended to multiple repositories for dataset building
- **Agent Tool Integration**: Demonstrated successful integration of program analysis tools with LLM agents using the LlamaIndex framework

4.2 Research Contributions

- **Metadata Extraction Framework**: Established a systematic approach to extracting program structure, dependencies, and call relationships from C codebases
- **RAG Foundation**: Built the infrastructure for retrieval-augmented generation by creating structured, queryable code metadata
- **Evaluation Platform**: Created a working prototype that demonstrates LLM capability to evaluate code metadata quality using structural context

4.3 Deliverables

The following artifacts have been produced:

- `build_map.py`: Complete static analysis pipeline (354 lines)
- `main.py`: LLM agent system with tool integration (118 lines)
- `code_structure.json`: Comprehensive structured dataset (15,000+ lines)
- Multiple SVG visualizations: Dependency and call graph representations
- Agent execution logs: Complete traces of LLM reasoning processes

5 Next Steps

The current implementation provides a solid foundation for the next phases:

1. **Dataset Expansion**: Scale the pipeline to process multiple popular GitHub repositories to build a large-scale evaluation dataset
2. **Comment Extraction**: Enhance the pipeline to extract and pair code comments with function definitions
3. **Quality Metrics**: Implement rubric-based scoring for comment completeness, accuracy, and consistency
4. **RAG Enhancement**: Integrate vector embeddings for semantic similarity search alongside structural retrieval
5. **Evaluation Framework**: Develop systematic evaluation protocols for measuring LLM performance in metadata quality assessment

6 Conclusion

This progress report demonstrates significant advancement toward the goal of using LLMs to evaluate code metadata quality. The implemented system successfully:

- Extracts structured program metadata from a production codebase
- Creates navigable representations of code relationships
- Deploys an LLM agent capable of intelligent code navigation
- Provides a foundation for scalable metadata evaluation research

The infrastructure is ready for expansion to larger datasets and more sophisticated evaluation tasks, positioning the project well for the research track investigating scalability of LLM-based metadata analysis and generation.