LLM Usage Appendix (CSE3063 Term Project)

Group: 13 **Project:** MiniRAG Chatbot (Iteration 1)

1. Overview of Usage

We utilized Large Language Models (specifically Google Gemini) primarily as a "Coding Assistant" and "Architecture Consultant" throughout the development of Iteration 1. The LLM was instrumental in adhering to SOLID principles, designing the Strategy Pattern interfaces, and refactoring heuristic algorithms. No direct code generation was used for the final logic without manual review and adaptation.

2. Key Prompts and Adaptations

Scenario A: Designing the Strategy Pattern (Architecture)

- **Prompt:** "Create a Java interface structure for a RAG pipeline that complies with the Strategy Pattern and Open/Closed Principle. I need interfaces for IntentDetector, Retriever, and AnswerAgent."
- **LLM Output:** Provided a basic set of interfaces (IntentDetector, Retriever) and a Controller class (RagOrchestrator).
- **Our Validation & Edit:** We accepted the interface names but modified the Retriever signature to accept KeywordIndex instead of a generic Map to better fit our domain model. We also ensured the RagOrchestrator used constructor injection for better testability, which wasn't present in the initial draft.

Scenario B: Refactoring Chunking Logic (Data Processing)

- **Prompt:** "My AnswerAgent cuts sentences in half because of dots in abbreviations like 'Prof.'. How can I split text better?"
- **LLM Output:** Suggested using a complex Regex lookbehind (?<=[.!?])\s+.
- **Our Validation & Edit:** We tested this but found it failed on list-based data (Course plans). We rejected the complex regex solution and instead pivoted to a "Paragraph-based" splitting approach as per the instructor's simplified requirements. We rewrote the IndexerMain to split by \R\R (double newlines) instead of sentences.

Scenario C: Enhancing Answer Selection Logic

- Prompt: "The standard sentence selection logic fails on list-based content like course schedules because the relevant information (e.g., 'Semester') is often in a header lines above the keyword match. Also, simple keyword counting is not enough for specific questions."
- LLM Output: Initially suggested a complex Regex-based lookbehind approach to capture headers, but this proved brittle across different document formats.
- Our Validation & Edit: We rejected the pure Regex approach for a more robust "Hybrid Extraction Strategy". We refactored the TemplateAnswerAgent to switch logic based on the content type:
    1. For Course Queries: We implemented a "Look-Back" mechanism that, upon finding a course code (e.g., CSE3063), scans upward to find the nearest "Semester/Term" header and includes it in the answer.

2. For General Queries: We maintained a heuristic scoring system but added "Context Expansion" to capture full blocks of text (e.g., keeping a professor's title and office together) instead of extracting single isolated lines. This ensures the answer preserves necessary context.

3. Code Correctness & Design Quality

- **Regex Validation:** All Regex patterns generated by the LLM (e.g., for extracting CSE\d{4}) were unit-tested against our specific corpus (ders_planı.txt) to ensure they didn't capture false positives.