Major Project Interim Report

Prakhar Singhal 2022111025
Raveesh Vyas 2022114002

October, 2025

1 Introduction

This report details the implementation of the end-to-end data preparation and ingestion pipeline for the Subject Matter Expert (SME) AI Agent project. The pipeline automatically ingests heterogeneous documents (PDFs, DOCX, etc.), enriches them with metadata, performs advanced hierarchical chunking to preserve context, generates domain-specific embeddings, and indexes them into a persistent vector database.

We have also implemented an automated batch ingestion script and a reranking model, making our system ready for the next phase of LLM integration and answer synthesis.

2 Dataset Description

This section details the corpus collected to train the SME agent.

2.1 Dataset Domain

Our chosen domain is **Software Engineering Principles** which includes system design and good coding practices.

2.2 Metadata Schema

As required, each document is automatically associated with the following metadata during ingestion to aid in filtering and context-awareness:

- `subject`: Inferred from the document's parent directory (e.g., "nlp_domain").
- `source`: The full file path of the document.
- `context`: The filename (e.g., "lecture_5_slides.pptx").
- `timestamp`: The "last modified" timestamp of the file.

3 Preprocessing & Chunking Strategy

This section justifies the critical technical decisions made for document processing.

3.1 Preprocessing Steps

Before chunking, all documents are passed through a preprocessing pipeline:

1. **Automatic Type Detection:** `SimpleDirectoryReader` (using `unstructured`) automatically detects and parses all required formats (PDF, DOCX, etc.).
2. **Deduplication:** A hash of each document's full text is generated. Any document with a duplicate hash is discarded, preventing redundant information.
3. **Normalization:** All text content is converted to lowercase.

3.2 Justification of Chunking Strategy

A core requirement was to select and justify an optimal chunking strategy.

Strategy	Pros	Cons
Fixed-Size	Simple to implement.	Arbitrarily splits text, often mid-sentence.
Content-Aware	Better semantic grouping.	Highly inconsistent chunk sizes. Poor for embedding.
Our Choice: Hierarchical	**Best of both.**	More complex to implement.

Table 1: Chunking Strategy Comparison

We implemented a **Hierarchical (Parent-Child) Chunking** strategy using `HierarchicalNodeParser`. This is the superior method for an SME agent as it solves the "lost context" problem.

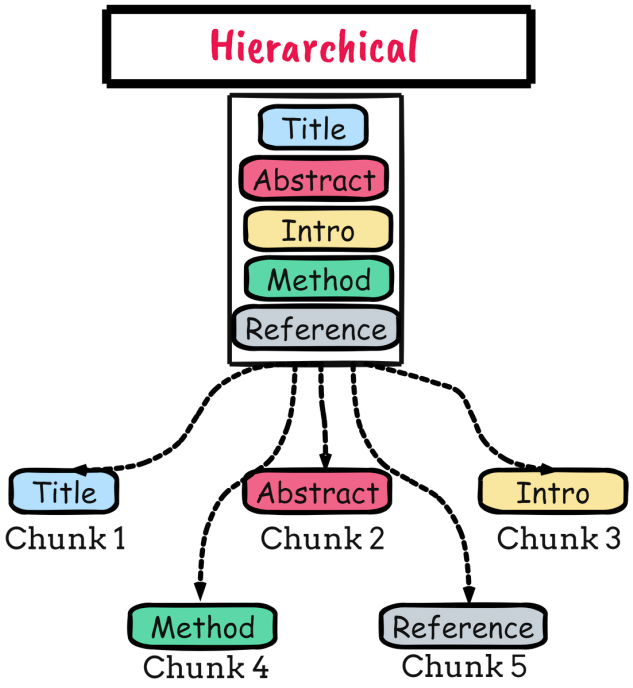

Figure 1: Hierarchical (Parent-Child) Chunking Strategy

3.2.1 Our Implementation

1. **Parent Chunks (2048 tokens):** A document is first split into large "parent" chunks. These represent a full "chapter" or section and preserve the broad context.
2. **Child Chunks (512 tokens):** Each parent chunk is then split into smaller "child" chunks. These dense, specific "pages" are used to generate embeddings.
3. **Third-Level (128 tokens):** A final, more granular split is available, though our primary strategy focuses on the 2048/512 relationship.

This ensures that when we retrieve a small, relevant *child* chunk, we can also fetch its *parent* node, giving the LLM the full context it needs to formulate an expert answer.

4 Embedding & Indexing Strategy

This section details the models and database used to store and retrieve knowledge.

4.1 Embedding Model Selection

We evaluated two models as required:

1. **Baseline:** `sentence-transformers/all-mpnet-base-v2`. This is a fast, effective all-rounder, but lacks depth for specialized academic or technical topics.
2. **Chosen Model:** `BAAI/bge-large-en-v1.5`. This 1.34 GB model was chosen as our "domain-specific" model. As a top performer on the MTEB (Massive Text Embedding Benchmark), it has a much more nuanced understanding of complex terminology.

4.2 Vector Database Selection

We selected **ChromaDB** as our vector database.

- It is open-source, runs locally without a server, and provides persistent storage (in the `./vector_db_store` directory). Its first-class integration with `llama-index` made it an ideal choice.

5 Bonus Features & System Architecture

5.1 [BONUS] Automated Batch Ingestion Pipeline

The script `data_pipeline/main_pipeline.py` serves as the automated batch ingestion pipeline. When executed, it automatically:

1. Scans the entire `source_documents` directory.
2. Performs all preprocessing and hierarchical chunking.
3. Generates embeddings for all new/updated nodes.
4. Adds them to the persistent ChromaDB.
5. Logs all operations and errors to `logs/ingestion.log`.

5.2 [BONUS] Two-Stage Retrieval with Reranking

A simple vector search can suffer from finding chunks that are only *semantically* similar but not *truly relevant*. We implemented a two-stage retrieval process to solve this.

1. **Stage 1: Vector Search (Fast):** The `BAAI/bge-large` model retrieves the Top-10 *candidate* child nodes.
2. **Stage 2: Reranking (Accurate):** These 10 nodes are passed to the `BAAI/bge-reranker-large` model (a Cross-Encoder) which re-sorts them by true relevance.

This ensures the final nodes passed to the LLM are the most accurate.