# Project Report

*LMA Major Project*

Course Name : Language Models and Agents
Course Code : CL3.410
Semester : Monsoon'25
Instructor Name : Prof. Vasudev Varma & Vandan Majudia

Pratyusha Mitra (2024701033)
(pratyusha.mitra@research.iiit.ac.in)

Kushagra Gupta(2025909001)
(kushagra.g@students.iiit.ac.in)

International Institute of Information Technology
Hyderabad, India

November 18, 2025

## Contents

# 1 Data Collection and Preparation

## 1.1 Phase 1: Document Collection and Organization

The document collection phase involved sourcing authoritative textual material relevant to the domain of **Food Safety and Nutrition**. A representative NCERT textbook, `12_homescience_eng_sm_2024.pdf`, was used as the initial data source to establish and test the complete question–answering pipeline.

All source documents were organized under a root directory within Google Drive. Each file was automatically detected and processed using the `PyPDFLoader` module from the `langchain-community` library. For every extracted document, a metadata schema was attached to facilitate structured indexing and retrieval. The metadata included attributes such as:

- **Subject:** Food Safety and Nutrition
- **Source:** NCERT Textbook
- **Timestamp:** ISO-8601 formatted ingestion time

This ensured that each chunk or retrieved passage could be traced back to its source context during later stages of retrieval and analysis.

The setup environment was configured in Google Colab, with the following dependencies installed:

```
!pip install -q langchain langchain-community langchain-openai
               sentence-transformers faiss-cpu pypdf
```

Google Drive was mounted to provide persistent storage and facilitate index reuse.

The project titled `food_safety_rag` was developed as a modular retrieval-augmented generation (RAG) system for the domain of **Food Safety and Nutrition**. Two primary data sources were used for corpus construction:

1. **NCERT Home Science Textbook (Class XII, 2024 edition):** extracted from `12_homescience_eng_sm_2024.pdf` using the `PyPDFLoader` utility.
2. **Hugging Face Dataset:** the publicly available corpus `yasserrmd/food-safety` was imported using the `datasets` library to supplement textbook material with contemporary food-safety-related documents.

All collected material was stored in a structured repository (`food_safety_rag/`) consisting of separate subdirectories for data, scripts, and configuration files. Each document was annotated with essential metadata fields, namely *subject*, *source*, and *timestamp*, to support traceability and provenance management. The repository follows the following layout:

```
food_safety_rag/
  data/
```

```
    food_safety_texts.txt
    combined_corpus.txt
    dataset.py
scripts/
    build_database.py
    retriever.py
    query_rag.py
    test_es.py
requirements.txt
docker-compose.yml
venv/
```

Dependency management was defined in `requirements.txt`, which included core libraries such as `transformers`, `sentence-transformers`, `datasets`, `pypdf`, and `elasticsearch`: contentReference[oaicite:1]index=1. All components were executed within a virtual environment and verified via shell scripts (`setup_rag.sh`, `build.sh`, `run.sh`) to ensure reproducibility.

## 1.2 Phase 2: Document Collection and Organization

- All source materials were collected under a root-level directory in Google Drive (`/food_safety_corpus/`). This directory contains heterogeneous files including PDF, DOCX, PPTX, TXT, and Markdown formats.
- The ingestion pipeline automatically detects document types and assigns appropriate loaders:
  - `PyPDFLoader` for PDF documents
  - `UnstructuredWordDocumentLoader` for DOCX files
  - `UnstructuredPowerPointLoader` for PPTX presentations
  - `TextLoader` for TXT files
  - `UnstructuredMarkdownLoader` for MD files
- Each document is enriched with metadata attributes to preserve provenance and structure:
  - **subject:** Food Safety and Nutrition
  - **source:** File name or dataset origin
  - **context:** Authoritative or supplementary reference
  - **timestamp:** Ingestion time in ISO format
  - **file_type:** Original document format
- The system integrates two complementary data sources:
  1. The NCERT textbook `12_homescience_eng_sm_2024.pdf`, processed with `PyPDFLoader`.
  2. The Hugging Face dataset `yasserrmd/food-safety`, imported through the `datasets` library to supplement the textbook content with real-world examples.
- A total of 748 pages were processed from the primary corpus, and all content was serialized into a unified document list with consistent metadata.

- The pipeline supports scalable addition of new authoritative or supplementary sources while maintaining consistent structure and traceability.

## 2 Preprocessing and Chunking

The preprocessing pipeline began by extracting raw text content from the PDF and applying a cleaning function to normalize case, remove extraneous symbols, and collapse redundant whitespace. All text was converted to lowercase and stripped of non-informative characters using regular expressions.

Document segmentation was performed using the `RecursiveCharacterTextSplitter` class from `LangChain`. This strategy allows content-aware chunking with adjustable granularity and overlap, maintaining semantic continuity across segments. The configuration used for this project was:

- `chunk_size = 1000` characters
- `chunk_overlap = 200` characters

This resulted in a collection of self-contained text chunks suitable for embedding and retrieval. The recursive splitter ensures that paragraph and sentence boundaries are respected wherever possible, reducing information fragmentation.

The data preprocessing pipeline was implemented both interactively (in Colab) and through the script `scripts/build_database.py`. It performs the following key operations:

1. **Text Normalization:** lowercasing, whitespace normalization, and removal of non-alphanumeric characters via regular expressions.
2. **Chunking Strategy:**content-aware segmentation using the `RecursiveCharacterTextSplitter` with parameters `chunk_size = 1000` and `chunk_overlap = 200`.
3. **Token-level Cleaning:** deduplication and filtering of empty or non-informative text segments.

The same logic was applied to both locally extracted text and data loaded from the Hugging Face dataset. All processed segments were concatenated into unified corpus files (`food_safety_texts.txt` and `combined_corpus.txt`) inside the `data/` directory. This ensured semantic continuity and minimized loss of contextual information between consecutive segments.

### 2.1 Preprocessing and Chunking

- Text preprocessing was applied uniformly to both the NCERT textbook and the Hugging Face dataset. Each document underwent normalization to remove noise, convert text to lowercase, and collapse redundant whitespace.
- The following regex-based cleaning operations were implemented:
  - Removal of URLs, HTML tags, and non-alphanumeric symbols.
  - Tokenization and lowercasing for consistent case handling.

  - Filtering of non-informative or very short segments.
  - Deduplication via MD5 hashing of cleaned text.
- The base chunking strategy adopted was **content-aware recursive splitting** using the `RecursiveCharacterTextSplitter`. Parameters were tuned as:
  - `chunk_size = 1000` characters
  - `chunk_overlap = 200` characters

  This ensured semantic continuity across paragraph boundaries.
- To facilitate hierarchical retrieval, **multi-granularity segmentation** was implemented using a token-based approach with the Hugging Face `bert-base-uncased` tokenizer:
  - `2048`, `512`, and `128`-token windows
  - Approximately 15% token overlap per level
- The pipeline was designed as a batch ingestion process (`process_documents_batch`) with progress tracking via `tqdm` and logging to `ingestion.txt`.
- All chunk levels (`content_recursive`, `lvl_2048`, `lvl_512`, `lvl_128`) were stored in a structured dictionary, later serialized for embedding.
- Comprehensive error handling and logging were implemented to capture ingestion anomalies and record per-stage statistics for reproducibility.

# 3 Embedding and Indexing

## 3.1 Embedding and Indexing

- A **parent–child hierarchy** was established to maintain document structure. Each source document (parent) was assigned a unique `parent_id`, and all derived text chunks (children) reference this identifier to preserve source linkage.
- Two embedding models were employed:
  - **General model:** `sentence-transformers/all-MiniLM-L6-v2` (384 dimensions) for baseline semantic coverage.
  - **Domain-specific model:** `allenai/specter2_base` (768 dimensions) tuned for scientific and educational corpora in food safety and nutrition.
- Embeddings were computed in optimized batches with GPU auto-detection and caching (`.npy` files) to minimize recomputation. Each embedding was normalized for cosine similarity search.
- Two FAISS vector indices were constructed:
  - **FAISS (General):** Built with MiniLM embeddings for fast, low-dimensional retrieval. [1]
  - **FAISS (Domain):** Built with SPECTER2 embeddings for domain-aware retrieval.

  Both indices were persisted in Google Drive under `/fsn_outputs/faiss_general` and `/fsn_outputs/faiss_domain`.
- Metadata for each chunk, including the parent identifier, granularity level, and file source, was serialized into `meta.pkl` files for structural integrity.
- A **retrieval and reranking module** was implemented:

  - Initial candidate retrieval from FAISS using L2 similarity.
  - **BGE Reranker (BAAI/bge-reranker-base)** applied for semantic relevance scoring.
  - Fallback to FAISS similarity if reranker is unavailable.
- The reranking step assigns contextual scores to query–document pairs, improving precision over base vector similarity.

**Bonus** A local hybrid retriever (FAISS + BM25) was also developed to emulate Elasticsearch-style mixed relevance ranking without requiring a running ES server.

The following code snippet summarizes the process:

```
model = SentenceTransformer("all-mpnet-base-v2")
embeddings = model.encode(texts, show_progress_bar=True, convert_to_numpy=True)
index = faiss.IndexFlatL2(embeddings.shape[1])
index.add(embeddings)
faiss.write_index(index, "faiss_food_safety.index")

with open("metadata.pkl", "wb") as f:
    pickle.dump((texts, metadatas), f)
```

Both the FAISS index and the metadata pickle file were stored in Google Drive for later reuse:

```
/content/drive/MyDrive/fsn_outputs/faiss_food_safety.index
/content/drive/MyDrive/fsn_outputs/metadata.pkl
```

## Verification

To validate successful indexing, a retrieval function was implemented to encode a user query, perform a similarity search against the FAISS index, and return the top-$k$ semantically closest chunks. For example, when queried with *"What are the main causes of food spoilage?"*, the system retrieved coherent and contextually relevant passages discussing factors such as microbial contamination, physical and chemical deterioration, and inappropriate storage conditions.

This confirmed that the embedding and indexing pipeline was functioning as intended and provided semantically meaningful retrievals for downstream question-answering tasks.

Semantic embeddings for each text chunk were generated using the `SentenceTransformer` model `all-mpnet-base-v2`. The embedding and indexing workflow was implemented both in the Colab prototype and in the script `scripts/retriever.py`, following these steps:

1. Compute dense 768-dimensional embeddings for all text chunks.
2. Build a FAISS index using the `IndexFlatL2` structure for efficient nearest-neighbor search.

3. Persist the vector index (`faiss_food_safety.index`) and its corresponding metadata (`metadata.pkl`) for later retrieval.

The following code summarizes the process:

```
model = SentenceTransformer("all-mpnet-base-v2")
embeddings = model.encode(texts, show_progress_bar=True, convert_to_numpy=True)
index = faiss.IndexFlatL2(embeddings.shape[1])
index.add(embeddings)
faiss.write_index(index, "faiss_food_safety.index")

with open("metadata.pkl", "wb") as f:
    pickle.dump((texts, metadatas), f)
```

The index and metadata were subsequently loaded from persistent storage using the `faiss.read_index()` and `pickle.load()` utilities. A retrieval function was defined in `retriever.py` and `query_rag.py` to encode user queries, perform top-$k$ similarity search, and return semantically closest text segments for downstream question answering.

### Verification

Initial evaluation queries, such as *"What are the main causes of food spoilage?"*, retrieved contextually accurate results covering microbial, chemical, and environmental causes of spoilage, confirming that the embedding and indexing pipeline was correctly implemented.

## 4 Core SME Capabilities

### Phase 2 Enhancement: Expanded Domain Corpus

In Phase 2, we significantly expanded the SME knowledge base by integrating two major authoritative datasets into the RAG corpus:

- **FSSAI Manuals and Guidelines** (India's Food Safety and Standards Authority)
- **WHO Food Safety Manuals and Training Documents**

These additions strengthened factual grounding, improved retrieval relevance, and enabled broader coverage of domain concepts such as hygiene protocols, contamination pathways, HACCP principles, food handling guidelines, and global best practices.

### 4.1 Capabilities Implemented

Our Subject-Matter Expert (SME) system for Food Safety is designed to support domain-grounded knowledge generation, multi-step reasoning, retrieval-augmented question answering, document creation, and e-mail delivery. The system initially used LangChain-based orchestration and was later we tried to extend with LangGraph to support multi-agent routing and workflow-level control absent in LangChain.

### 4.2 Capabilities Implemented

- **Expert Content Generation:** The system can produce structured domain content such as one-page summaries, SOP notes, food safety explanations, quizzes, and instructional modules. Outputs can be exported as PDF, DOCX, or slides.
- **Adaptive Explanations & Reasoning:** The system provides step-by-step reasoning for complex food safety concepts (e.g., cross-contamination, hygiene workflows, contamination pathways). It supports follow-up questions using local memory.
- **Self-Learning via Feedback:** A human feedback memory module (`feedback_store.py`) stores user ratings and comments, enabling adaptive refinement of explanations.
- **Automated Content Delivery:** The agent can generate a report, format it as a PDF, and send it by e-mail using SMTP credentials stored in `.env`. This demonstrates end-to-end task execution beyond chat-based QA.

## 5 Agent for Chat, Planning, Reasoning, and Routing

To meet the requirement of robust task routing and multi-step planning, we implemented an agent capable of:

- **Conversational Planning:** The agent decomposes user intent into subtasks such as retrieval, summarization, tool invocation, and document export.
- **Reasoning-Aware Routing:** Using LangGraph, we constructed a directed workflow where nodes represent LLM reasoning, retrieval decisions, tool usage, and final answer synthesis. LangGraph enables conditional branching, retry logic, and memory-aware context routing, which was not feasible using LangChain alone.
- **Context and Memory Management:** Conversation history, top-K retrieved context, and user-specific metadata are combined using structured messages and a hybrid RAG prompt template.
- **Error-Aware Execution:** The agent detects tool failures (e.g., DOCX generation failure) and switches to alternative strategies (PDF generation fallback).

## 6 LLM Models and Retrieval-Augmented Generation (RAG)

### 6.1 LLM Usage

We experimented with Gemini API key. Key techniques included:

- carefully engineered system prompts;
- few-shot prompting for domain-specific answer formatting;
- iterative prompting to refine suboptimal model behaviour;
- using a shared pre-trained model across both QA and report generation tasks.

We documented systematic behaviour across different models:

- Gemini produced more reliable factual grounding in food safety.
- Smaller models struggled with multi-step reasoning and hallucinated procedural details when RAG context was insufficient.

### 6.2 Hybrid Retrieval and Relevance Ranking

A hybrid retrieval pipeline was built using Elasticsearch 8.11.1, combining:

- **Dense vector search** via Sentence Transformers (all-mpnet-base-v2);
- **Keyword search** (BM25);
- **Hybrid scoring** combining cosine similarity, BM25 score, and metadata-based reranking (document type, section, domain).

To ensure robustness:

- multi-level chunking (2048 → 512 → 128 tokens) & hierarchical chunking;
- fallback to LLM-only answer generation when Elasticsearch is down;
- manual evaluation of retrieval quality using `query_rag.py`.

## 7 Tool capabilities

The `agent/tools.py` module integrates capabilities required by the assignment:

- **Knowledge Retrieval:** RAG pipeline accessible as a callable tool.
- **Document Generation:** PDF (via ReportLab), DOCX, and PPTX generation for quizzes, summaries, or instructional material.
- **Email Automation:** SMTP-based email sending with attachments, enabling workflows such as “Generate one-page food safety note → Export as PDF → Email to user”.
- **Error Handling:** Automatic fallback strategies and retries for tool failures.

## 8 System Components & Architecture

The folder structure of the system is as follows which is used in langchain:

```
/SME/
 .env
```

```
requirements.txt
docker-compose.yml      # Elasticsearch
workflow.py             # CLI orchestrator
data/                   # All domain documents
rag/
   retriever.py        # BM25 + Dense + Hybrid
   query_rag.py
scripts/                # Ingestion + indexing
agent/
   agent.py            # Planner + Router + Reasoning
   main_api.py         # FastAPI server
   feedback_store.py
   tools.py
reports/outputs/
```

## 8.1 API and Server Design

- **FastAPI Server:** Handles chat, tool calls, planning, and workflow execution. Supports asynchronous calls and JSON schema validation.
- **Chat/Tool/Agent Modules:** Each invocation goes through a router that determines whether to call:
  1. RAG pipeline,
  2. LLM reasoning,
  3. or a tool (PDF, DOCX, email).
- **LangGraph Workflow:** Replaces traditional LangChain agents to allow explicit state transitions, deterministic planning, retry loops, and multi-agent behaviour.

```
/SME/
 .env
 requirements.txt
 docker-compose.yml            # Elasticsearch service

 workflow.py                   # Old CLI orchestrator (Phase 1)
 workflow_client.py            # New LangGraph workflow executor (Phase 2)

 data/                         # All domain documents
    FSSAI_PDFs/
    WHO_Manuals/
    text_data/
    Textbooks/

 rag/
    retriever.py              # BM25 + Dense + Hybrid Retrieval
    query_rag.py              # Manual retrieval testing

 scripts/                      # Ingestion + indexing utilities
```

```
      build_database.py
      generate_metadata.py
      tests/

   agent_v2/                          # Phase 2 | LangGraph Agent System
      main_api.py                   # FastAPI server (chat, workflow, planning)
      agent_v2.py                   # High-level router + API integration

      graph/                        # LangGraph workflow implementation
         workflow_graph.py         # Full workflow assembly (graph definition)
         state.py                  # State object for graph execution
         nodes/
             classify.py           # Classifier node (identify task type)
             plan_node.py          # Planning node for multi-step tasks
             rag_node.py           # Retrieval node calling hybrid RAG
             quiz_node.py          # Quiz generation node
             report_node.py        # Report generation node (PDF/DOCX/PPT)
             tool_nodes.py         # Execution nodes for tool invocations
             execute_step_node.py # Step executor + error recovery

      mcp/                          # Tool servers (Modular Command Protocol)
          docx_server.py            # DOCX generation service
          pdf_server.py             # PDF generation service
          ppt_server.py             # PPTX generation service
          email_server.py           # Email sender service
          search_server.py          # Search/RAG entry-point for tools

   reports/
       outputs/                       # Auto-generated outputs (never committed)
           metadata.csv
           workflow_output.pdf
           workflow_output.docx
```

## 9 Testing and Validation

We tested the pipeline using reproducible CURL commands:

- **Test 1: Basic QA** – cross contamination.
- **Test 2: Follow-up QA** – checking conversational memory.
- **Test 3: Non-RAG query** – model-only knowledge.
- **Test 4: RAG failover** – disabling Elasticsearch to test fallback behaviour.

These tests confirmed the system's stability and its ability to degrade gracefully.

## 10 Learning Process: Strategies, Failures, and Adaptations

### 10.1 Successful Strategies

- Strict chunk-level metadata improved retrieval precision significantly.
- LangGraph-based routing resolved ambiguity in multi-step workflows.
- Structured JSON prompts improved accuracy of tool outputs.
- Feedback memory improved clarity and accuracy over multiple turns.

### 10.2 Observed Failures

- Small LLMs sometimes hallucinated food safety procedures when RAG context was insufficient.
- DOCX generation failed for long texts, requiring a PDF fallback.
- LangChain agents struggled with deterministic workflow execution (looping behaviours, unclear routing), necessitating the move to LangGraph.
- Dense-only retrievers retrieved semantically similar but contextually irrelevant content (e.g., hygiene in healthcare vs. food).

### 10.3 Adaptations

- Introduced hybrid BM25 + dense retrieval with metadata filters.
- Added explicit “verification nodes” in LangGraph to confirm correctness before tool invocation.
- Implemented a robust fallback chain: DOCX $\rightarrow$ PDF.
- Introduced reranker scoring to reduce semantic drift.

**N.B. Demonstration Video** A complete end-to-end demonstration video of the SME System (Phase 2: Langchain working, LangGraph Workflow, MCP Tooling, RAG Integration, and Report Generation) has been prepared and uploaded to the Google Drive.

**Demo Video Link (to be updated):** `https://drive.google.com/drive/folders/1QaonM1DRrIJYaH6KuJfLMctWfy30lETS?usp=share_link`

## References

[1] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, “The faiss library,” 2024.