# Distributed RAG-Based Web Scraper Framework — Midterm Report

Date: October 28, 2025

Student: [Your Name Here]

Course: Distributed Systems

Instructor: [Instructor Name]

## 1. Objective

The goal of this project is to design and implement a distributed web scraping system that collects, processes, and indexes website content to support Retrieval-Augmented Generation (RAG). The system integrates distributed task execution, message queuing, data cleaning, semantic indexing, and an API layer for data retrieval and question answering.

## 2. System Architecture Overview

The system consists of several core components working together in a distributed pipeline:

1. RabbitMQ Queue – Manages URLs and distributes scraping tasks.  
2. Ray Workers – Execute Scrapy spiders concurrently to collect HTML pages.  
3. MongoDB – Stores both raw HTML and cleaned text data.  
4. Cleaner – Processes raw HTML to extract meaningful plain text.  
5. FAISS Index – Converts text into embeddings and stores them for semantic search.  
6. FastAPI Server – Provides endpoints for viewing raw data, searching, and RAG-based question answering.

## 3. Data Flow Summary

1. URLs are pushed to the RabbitMQ queue.  
2. Ray workers consume these URLs, scrape them using Scrapy, and insert the raw HTML into MongoDB.  
3. The cleaner script removes scripts, styles, and tags, and writes cleaned text back into MongoDB.  
4. The FAISS indexer encodes each cleaned document into vector embeddings using Sentence Transformers.  
5. The FastAPI layer retrieves documents or performs RAG-style queries using semantic similarity search.

## 4. Example Data Transformations

Below is an illustration of how data transforms through the pipeline:

Raw HTML (stored in raw\_html):

<html><body><div class='quote'>“The world as we have created it is a process of our thinking.” by Albert Einstein</div></body></html>

Cleaned Text (stored in clean\_text):

Quotes to Scrape The world as we have created it is a process of our thinking. by Albert Einstein

Embedding Representation (stored in FAISS):

[0.123, -0.045, 0.682, ..., -0.117]

## 5. Environment Setup & Execution

1. Start RabbitMQ and MongoDB using Docker:  
 docker run -d --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:management  
 docker run -d --name mongodb -p 27017:27017 -v mongo-data:/data/db mongo:6  
  
2. Create and activate a virtual environment, then install dependencies:  
 python3 -m venv myenv  
 source myenv/bin/activate  
 pip install -r requirements.txt  
  
3. Start Ray and the scraper workers:  
 ray start --head  
 python scraper/ray\_workers.py --queue-url amqp://guest:guest@localhost// --sites-file infra/sites.txt  
  
4. Clean data and build embeddings:  
 python processing/cleaner.py  
 python rag/index.py --rebuild  
  
5. Run the FastAPI service:  
 uvicorn api.main:app --reload

## 6. Example API Queries

• GET /raw – Retrieve stored raw HTML documents.  
• GET /search?q=keyword – Search through cleaned documents using semantic similarity.  
• POST /rag – Perform a retrieval-augmented query using embeddings as context.

## 7. Results and Screenshots

Include full-screen screenshots of each phase:  
1. Environment setup (terminal + Docker running)  
2. Distributed scraping (Ray workers logs)  
3. Data cleaning (terminal output)  
4. Index building (FAISS creation)  
5. FastAPI running (Swagger UI and example queries)

## 8. Discussion

This system successfully demonstrates a distributed data processing architecture using Python. Ray ensures scalability and parallel scraping. RabbitMQ decouples data sources from processing tasks. MongoDB serves as a simple document store for both raw and cleaned data. FAISS enables high-speed semantic search, while FastAPI offers an accessible interface for integration and testing.

## 9. Conclusion

The project meets the objectives of implementing a distributed scraping and retrieval system. It integrates multiple technologies cohesively and demonstrates practical knowledge of distributed systems, data pipelines, and modern AI retrieval techniques.