AI-Driven Enhancements for Network Management Systems in Subsea Cable Infrastructure

Gavin Coates, Leo Kopitzko, Matthew McDowall, Philip Workman, Prince Ojha

Department of Computer Science, University of New Hampshire, Durham, NH 03824

Sponsor: Yunlu Xu, Director of Software Development, SubCom

Abstract

Submarine cable systems carry over 95% of intercontinental data traffic, and the ITU-T G-series standards that govern them span hundreds of pages of dense technical specifications. Engineers lose significant time locating information buried across these documents. We built a fully local AI knowledge assistant that uses Retrieval-Augmented Generation to answer natural-language questions about subsea cable systems with inline citations back to the source recommendations. The system parses ITU-T PDFs with Docling, embeds semantically-chunked passages into ChromaDB via nomic-embed-text, and grounds Qwen3-4B responses through a two-stage retrieve-and-re-rank pipeline. A React 19 frontend with client-side citation injection delivers traceable answers — all running on consumer hardware with zero external API calls.

Introduction

Context & Background

- Submarine cables carry >95% of intercontinental data traffic; SubCom has deployed 200+ systems worldwide since 1955.
- The governing ITU-T G-series standards (G.972, G.973, G.975, G.975.1) span hundreds of pages of dense technical content.

Problem

- Engineers and NMS operators spend excessive time locating specific information across multiple lengthy standards.
- Keyword search misses semantic relationships; manual cross-referencing is slow and error-prone.

Vision & Goal

- A locally-hosted AI assistant that answers natural-language questions with accurate, source-cited responses.
- Every component runs on the user's machine — ensuring data privacy and zero inference cost.

Success Criteria

- Grounded answers with inline ITU-T citations
- Two-stage retrieval with cross-encoder re-ranking
- Multi-turn chat with persistent history
- Fully offline operation after setup
- Runs within 4 GB RAM on consumer hardware

Requirements

Functional Requirements

- **Natural-Language Q&A:** Users ask questions in plain English and receive accurate, contextual answers grounded in the ITU-T corpus.
- **Source Citations:** Every answer cites specific recommendations and sections (e.g., G.975.1 § 3.2.1) with clickable source cards.
- **Retrieval & Ranking:** The system retrieves and re-ranks relevant passages before generation.
- **Multi-Turn Chat:** Thread-based conversations with persistent history across sessions.
- **Document Ingestion:** New PDFs can be added to the knowledge base through the UI.

Non-Functional Requirements

- **Privacy:** All inference and storage run locally — no external API calls.
- **Latency:** Query-to-answer under 15 s on consumer hardware (measured 5–12 s warm).
- **Accuracy:** Answers grounded in retrieved context, not hallucinated.
- **Footprint:** Operates within ~4 GB RAM and ~2 GB disk using quantized Qwen3-4B (Q4_K_M).
- **Portability:** Runs on macOS and Windows via Python, Node.js, and Ollama.
- **Extensibility:** Swapping the LLM, embedding model, or adding documents requires minimal code changes.

Design

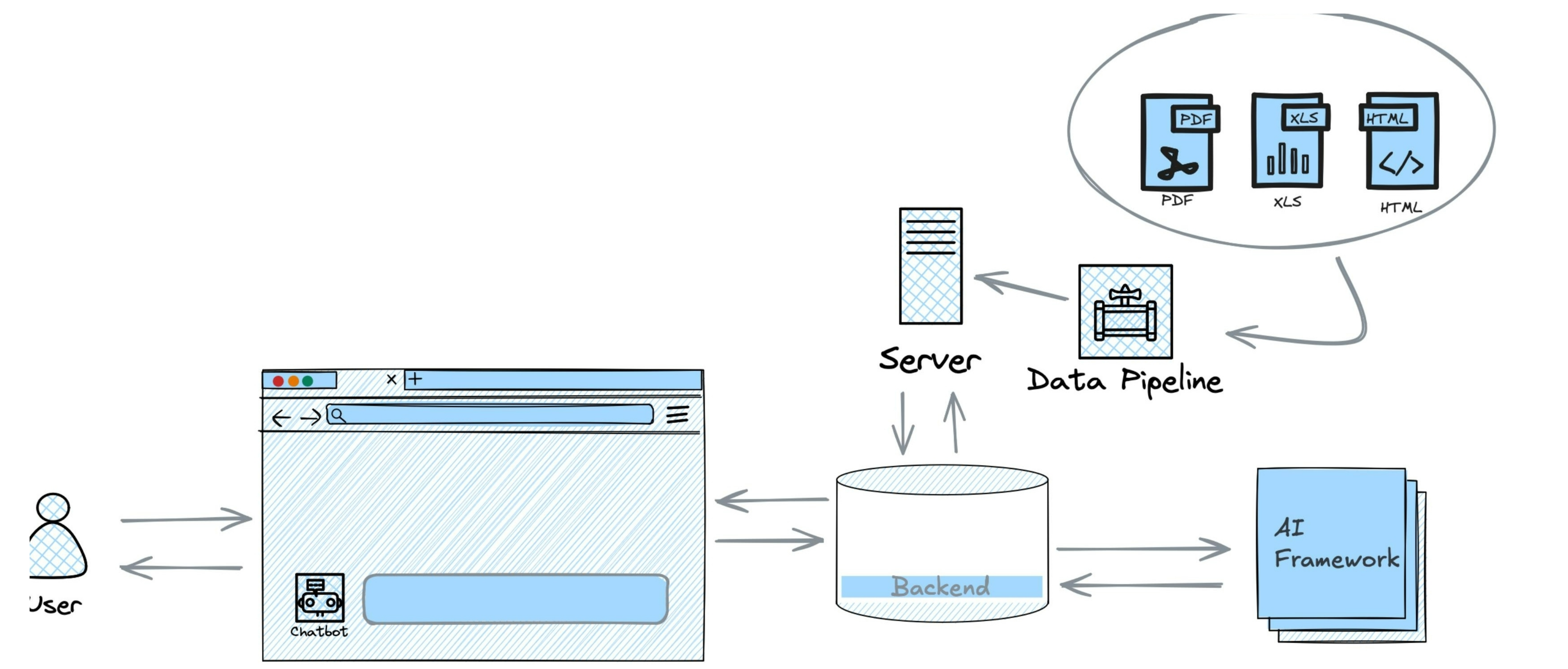

Figure 1 — System architecture. User interacts with Chatbot UI, which communicates with Backend and AI Framework. Data Pipeline ingests documents (PDF, XLS, HTML) into the Server.

Local stack — runs entirely on consumer hardware:

- **React 19 + TypeScript + Tailwind** — chat UI with Zustand state, thread persistence, inline citation badges
- **FastAPI + Uvicorn** — REST API serving both /query endpoint and the React SPA
- **LangChain + Ollama** — orchestrates RAG pipeline with Qwen3-4B (Q4_K_M) for generation
- **ChromaDB + nomic-embed-text** — persistent local vector store with semantic embeddings

Query Processing Flow

1. **User Query** — Natural language input via React chat interface
2. **Over-Retrieve** — ChromaDB cosine similarity search, k=20 candidates
3. **Re-Rank** — Cross-encoder (ms-marco-MiniLM) scores each pair → top 5
4. **Build Context** — Top 5 chunks formatted with [doc — heading] labels
5. **Generate** — Qwen3-4B produces cited response grounded in context

Key Design Decisions

- Two-stage retrieval (dense k=20 → cross-encoder top-5) promotes truly relevant chunks over tangential matches.
- Hierarchical chunking preserves section breadcrumbs so every chunk carries its heading context for the LLM and source cards.
- Client-side citation injection scans answers for ITU-T IDs and inserts clickable badges — decoupling citations from LLM output.
- Fully local execution (Ollama + ChromaDB) ensures data sovereignty with no external API calls.

User Interface

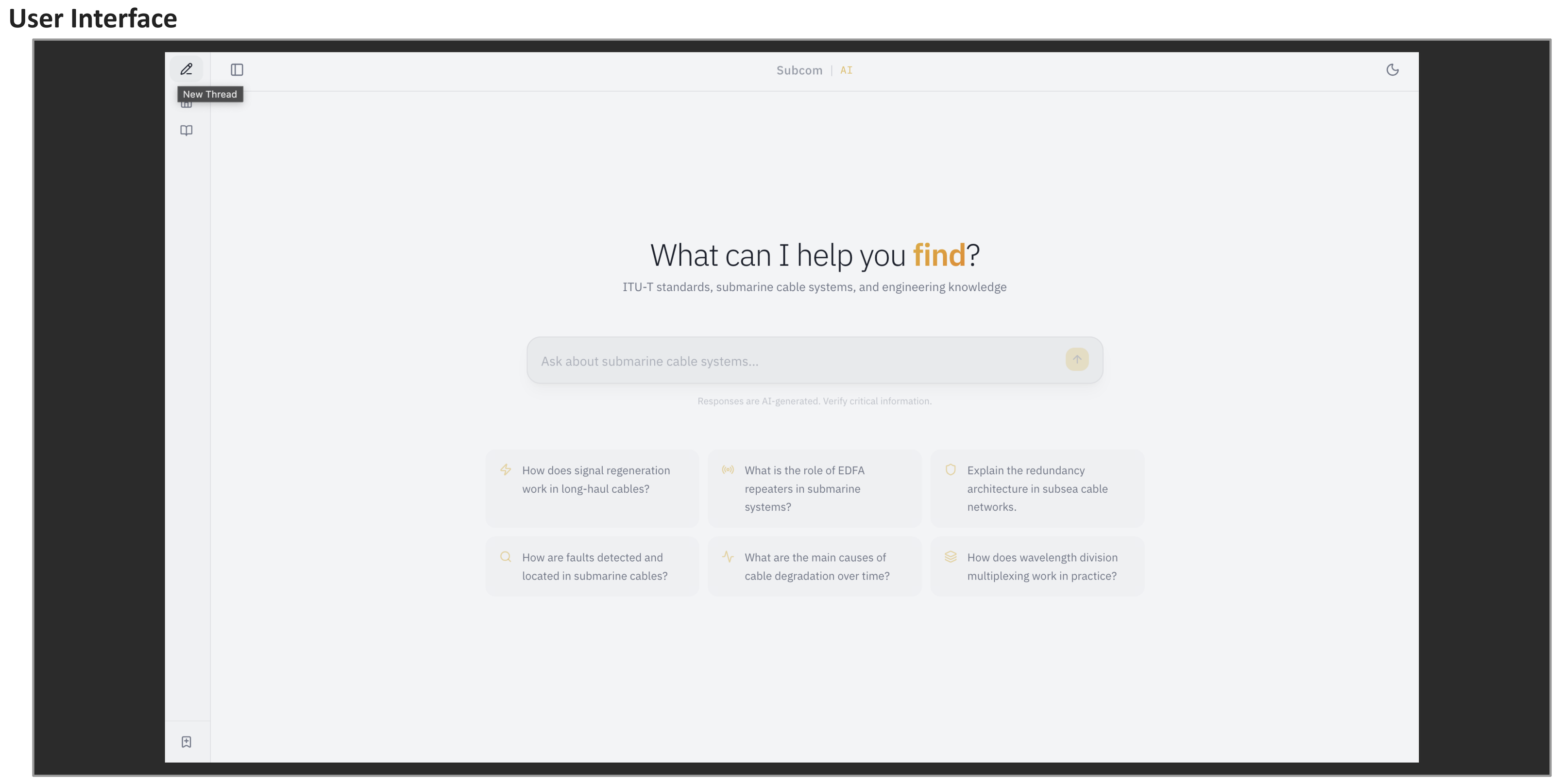

Figure 2 — React chat interface with thread sidebar, inline citation badges, and source cards linking back to ITU-T recommendations.

Implementation & Testing

Agile Development (4 Sprints)

- Sprint 1 — Research: RAG architecture, sponsor requirements, model benchmarking
- Sprint 2 — Setup: Ollama deployment, ChromaDB init, Docling pipeline
- Sprint 3 — Development: RAG pipeline, FastAPI backend, React frontend
- Sprint 4 — Integration: ITU-T ingestion, citation system, testing

Document Corpus Processed

- G.972, G.973, G.975, G.975.1 — ~218 pages total
- ~350 semantically-chunked passages embedded into ChromaDB
- Hierarchical chunker preserves section headings and cross-references

How Tools Are Used

- **Docling** parses ITU-T PDFs with structural awareness — preserving headings, tables, and figures via HierarchicalChunker.
- **Ollama** serves Qwen3-4B locally via REST; handles GGUF quantization natively on CPU or modest GPU.
- **LangChain** orchestrates the RAG chain: embed → retrieve → re-rank → inject → generate.
- **ChromaDB** stores nomic-embed-text embeddings locally with cosine similarity for fast recall.

Testing Results

- Vector search <100 ms; re-ranking ~1–2 s; full query 5–12 s warm.
- Cross-encoder consistently promotes directly-relevant chunks (4–5 of top 5) over tangential matches.
- Deterministic generation (T=0.0) gives reproducible evaluation.
- Memory footprint: ~2–4 GB RAM, within target.

Evaluation & Conclusions

Results Against Success Criteria (MOV)

Criterion	Result
Grounded Q&A with citations	Met
Two-stage retrieval pipeline	Met
Multi-turn chat + persistence	Met
Latency <15 s on consumer HW	Met
Fully local — no external APIs	Met

- We delivered a fully local, privacy-preserving RAG assistant that answers ITU-T questions with grounded, cited responses.
- The two-stage retrieval + hierarchical chunking architecture produces measurably more relevant context than naive similarity search alone.

Limitations

- Corpus currently limited to four ITU-T recommendations.
- Responses are returned in full rather than streamed token-by-token.
- Occasional over-reliance on general knowledge when context is sparse.

Future Work

- Expand corpus to full ITU-T G-series and beyond
- Server-sent event streaming for perceived latency
- Hybrid dense + BM25 retrieval for exact technical terms
- LLM-generated follow-up questions; real backend progress events

Acknowledgments

We would like to thank our project sponsor **Yunlu Xu, Director of Software Development at SubCom**, for providing project guidance, infrastructure access, and technical documentation. We also thank the **UNH Department of Computer Science** and the CS-IT 792 Capstone Program for their support and guidance throughout this project.