Project Proposal: Personal RAG-Augmented LLM Assistant for CTF Problem-Solving

1 Project Definition & Research Question

Problem: During Capture-the-Flag (CTF) competitions I must recall exploitation techniques scattered across blog posts, GitHub write-ups, and forum threads. Searching by hand is slow and incomplete.\

Research question: *If I pair a lightweight Retrieval-Augmented Generation (RAG) repository of public write-ups with a large-language model (Gemini 2.5 Flash), does the assistant become noticeably more helpful to me on unseen CTF challenges than the same LLM running without external context?*\

Relation to class: The work touches every stage discussed in CS 210, including data collection, storage, integration, transformation, analysis, and visualization, using real-world tooling instead of toy datasets.

2 Personal Importance

- A well-indexed RAG store can hold far more structured knowledge than any individual can memorise, while semantic retrieval lets me ask natural questions instead of crafting keyword searches.
- The assistant is designed strictly for **my private workflow**, optimised for speed, relevance, and privacy; if it proves useful it will become my long-term "second brain" for security competitions.

3 Technical Plan

Data collection. I will fetch challenge metadata through the *ctftime.org* REST endpoint and curate roughly 30 Markdown or PDF write-ups that cover web-exploitation, cryptography, and reverse-engineering. Retrieval uses Python 3.12 with requests and is orchestrated via a simple Makefile and Git for version control.

Structuring with LLMs. A Cloud Run microservice (Docker + Poetry) sends each raw write-up to **Gemini 2.5 Flash** on Vertex AI, prompting it to emit canonical JSON fields: question, steps_taken[], resolution, category, difficulty, and tags[]. Responses are validated with pydantic models before storage.

Storage layer. Structured documents land in **MongoDB Atlas** (ctf_writeups collection). I will exploit MongoDB's flexible schema, full-text index for fallback keyword search, and built-in TTL indexes for easy corpus refresh.

Embedding & vector index. Each document is chunked to $\leq$ 1 K-token segments with tiktoken. Embeddings are produced by **gemini-embedding-002** via the google-cloud-aiplatform SDK and written to **Vertex AI Vector Search**. Embedding IDs are back-linked in MongoDB for traceability.

Query pipeline. A lightweight **FastAPI** CLI tool accepts my natural-language question, retrieves the top-k ($\leq$ 5) most similar chunks from the vector index, and passes them, along with the query, to Gemini 2.5 Flash. The model returns a concise hint plus citations. The entire stack is containerised and deployed on Cloud Run, with Google Cloud Scheduler triggering weekly re-embeddings.

Evaluation protocol. For ten unseen challenges I will run two modes:

1. **No-RAG:** ask Gemini 2.5 Flash directly.
2. **RAG:** ask through the pipeline. Each answer is scored 1–5 for usefulness, and I will record minutes-to-flag when applicable using a simple Jupyter notebook.

4 Data & Access

- **Public API:** CTFtime JSON feed (events, tasks, URLs).
- **Manual corpus:** hand-selected write-ups ($\approx$ 30 documents, $\leq$ 50 MB).

5 Models, Algorithms & Tooling

- **LLM structuring & generation:** Gemini 2.5 Flash (Vertex AI, 128k context).
- **Semantic retrieval:** Cosine similarity over 3 072-dim gemini-embedding-002 vectors (Vertex AI Vector Search).
- **Programming stack:** Python 3.12, Poetry for dependency management, Docker, FastAPI, PyMongo, google-cloud-aiplatform, and JupyterLab for analysis.

6 Hypothesis & Evaluation Metrics

- **Hypothesis.** The RAG-augmented assistant will (a) score at least one full point higher (on a 1–5 scale) for usefulness than the No-RAG baseline **and** (b) shorten my average solve-time by $\geq$ 20 % across the ten-challenge test set.
- **Primary metric.** Mean usefulness score.
- **Secondary metrics.** Mean minutes-to-flag (when solved) and median retrieval latency.
- **Success criterion.** Hypothesis holds on $\geq$ 60 % of test challenges.

7 Risks & Mitigations

- **Extraction errors.** Prompt iteration and manual spot-checks during initial ingestion.
- **Sparse corpus.** Focus evaluation on *helpfulness*, not exhaustive coverage; corpus can be expanded incrementally.
- **Budget constraints.** Maintain < 5 K embeddings; batch LLM calls and exploit Vertex AI's free-tier quotas.