Final Project Report

Title: Gap Analysis Agent: Built with RAG, LLMs, and Real-World Compliance Controls

Subtitle: Includes CMMC Controls. Adaptable to ISO 27001, NIST 800-53, and global GRC frameworks.

1. Introduction

This report documents the design and implementation of a Gap Analysis Agent tailored to assist cybersecurity documentation against compliance standards, particularly focusing on the Cybersecurity Maturity Model Certification (CMMC). The agent is an orchestrated pipeline composed of modular components that simulate a compliance gap assessment process. By integrating vector search with Retrieval-Augmented Generation (RAG) and a local Large Language Model (LLM), the system performs automated reasoning, control mapping, and reporting based on cybersecurity policy documents.

Built with offline capability and explainability in mind, this tool demonstrates how lightweight AI systems can emulate auditor-level assessments using open-source infrastructure.

2. Folder Structure Overview

```
gap-analysis-agent-cmmc/
├── controls/            # sample_controls.json — 10 CMMC controls
├── data/        # sample_policy.txt —  policies simulating real-world practices
├── embeddings/          # Vector chunks created from embedded policies
├── embed_engine.py       # Convert policy chunks into embeddings using MiniLM
├── export_gap_to_csv.py  # Helper to generate CSV report from JSON results
├── main.py              # Input validator to ensure format and integrity before processing
├── outputs/             # Final reports (gap_report.json, CSV exports)
├── parser/              # Optional pre-processing utilities (currently unused)
├── rag_engine.py         # Phase 3: Retrieve+reason logic for gap detection using LLM
├── README.md            # Full project instructions, methodology, and setup
├── requirements.txt      # Python dependencies for the local environment
├── run_dashboard.bat     # Launcher script for Streamlit UI
├── ui/                # Streamlit frontend interface (streamlit_app.py)
├── vector_db/           # Persistent ChromaDB store for vector search
├── venv/                # Local Python virtual environment
└── .gitignore           # Version control exclusions
```

3. Methodology & Workflow

The system is structured into four key phases:

Phase 1: Input Preparation

- Goal: Construct and format the input policy and control datasets.
- Sample policy constructed to reflect real-world cybersecurity readiness practices
- 10 representative controls selected from CMMC Levels
- Controls are stored in JSON format with control_id, description, and level

Phase 2: Embedding Engine (``)

- Goal: Convert policy chunks into embeddings using a lightweight transformer model.
- Text split into 36 chunks (~500 tokens each)
- Embeddings created using all-MiniLM-L12-v2 (HuggingFace)
- Stored in ChromaDB under collection policy_control_embeddings
- *Why MiniLM?*: Its speed and small size make it ideal for edge environments.

Phase 3: Retrieval + Reasoning (``)

- Goal: Match controls against embedded policies using semantic similarity and LLM-based reasoning.
- Retrieves the top 3 policy chunks per control using cosine similarity
- Uses local LLM (gemma:2b via Ollama) to reason per control
- Outputs structured JSON:

```
{
 "control_id": "AC.L1-3.1.1",
 "status": "Partially Met",
 "matched_text": "All system users must authenticate using MFA...",
 "confidence_score": 0.383,
 "explanation": "MFA is present, but user authorization roles are not clearly defined."
}
```

- *Why Gemma 2B?*: Delivers balanced reasoning with minimal hardware (8 GB RAM).

Phase 4: UI Dashboard (``)

- Goal: Provide a user-friendly interface to run, filter, and export analysis results.
- Local interface to:
  - View policy/control files
  - Trigger full gap analysis
  - Filter by match status or control level
  - Export CSV report
- Hosted locally at: http://localhost:8501

4. **Sample Execution Output**

Embedding Run Log:

- Created 36 policy chunks
- Stored in ChromaDB (collection: policy_control_embeddings)

RAG Comparator Log:

- Loaded 10 controls
- Evaluated with gemma:2b LLM

Output Status:

- Fully Met: 0
- Partially Met: 7
- Not Met: 3

Saved as: outputs/gap_report.json

CSV Export: Exported to: outputs/gap_report_20250624_2143.csv

5. **Results Summary**

Status	Count	Percentage
Fully Met	0	0%
Partially Met	7	70%
Not Met	3	30%

While foundational practices are present, higher-level requirements lack depth in policy documentation.

6. **Key Strengths**
   - **Offline and Self-Contained** – no cloud dependencies, API keys, or internet required
   - **Explainable and Transparent** – context, reasoning, and score provided for every control
   - **Minimal Hardware Requirement** – optimized for 8 GB RAM, 4 GB VRAM systems
   - **Modular and Maintainable** – independent phases simplify debugging and customization
   - **Adaptable Framework** – extendable to NIST 800-53, NIST CSF, ISO 27001, and DPDP like frameworks

7. **Future Roadmap**

   To support ongoing development and scale, the following roadmap outlines actionable enhancements for future versions of the Gap Analysis Agent:

   - **Bulk Upload Support**: Enable CSV/YAML-based control ingestion to allow rapid onboarding of larger control datasets.
   - **Smarter PDF Parsing**: Integrate PyMuPDF to accurately extract content from scanned or semi-structured compliance documents.
   - **Role-Based UI**: Design a segmented dashboard experience tailored for CISOs, analysts, and auditors to manage access and visibility.
   - **Larger Model Support**: Experiment with models such as Mistral 7B and Gemma 7B to improve semantic accuracy and reasoning depth.
   - **Audit Trail Generator**: Export detailed justifications for each control decision, creating a tamper-evident trail for formal audit submission.

8. **Conclusion**

   The Gap Analysis Agent demonstrates that audit-grade compliance analysis can be both local and explainable. It bridges the gap between lightweight tooling and regulatory alignment for small to mid-sized organizations.

   This project proves that localized, transparent, and technically sound compliance automation is within reach. It delivers not just checklist matching, but structured reasoning that mirrors the expectations of real auditors.

Version: 1.0 — June 2025