AI Hallucination Detection System - Project Plan (First Draft)

# Overview

This project aims to build an AI hallucination detection and correction system using Gemini Pro as the base LLM, TruthfulQA as the benchmark dataset, and Wikipedia + ChromaDB as the retrieval knowledge base. The system will detect hallucinations in model outputs, correct them using grounded evidence, and present users with fact-based responses along with confidence scores and citations.

# Timeline (6 Weeks)

* Week 1–2: Setup Gemini SDK, LangChain, TruthfulQA dataset, and retrieval pipeline (Wikipedia + ChromaDB).
* Week 3–4: Implement hallucination detection (semantic similarity + NLI) and self-correction pipeline.
* Week 5: Build CLI, Flask backend, Streamlit frontend, and optional REST endpoints.
* Week 6: Testing, benchmarking (TruthfulQA), visualization of results, and final documentation/report.

# Team Roles & Responsibilities

## Member 1: Data & Retrieval Engineer

Responsibilities:  
- Load TruthfulQA dataset from Hugging Face using Python `datasets` library.  
- Implement Wikipedia API calls for real-time evidence retrieval.  
- Setup ChromaDB to store a subset of Wikipedia documents.  
- Use Hugging Face sentence-transformers (e.g., `all-MiniLM-L6-v2`) to generate embeddings.  
  
Preferred Technology:  
- Python, Hugging Face Datasets, Wikipedia API, ChromaDB, sentence-transformers.  
  
Beginner-Friendly Steps:  
1. Install libraries: `pip install datasets wikipedia chromadb sentence-transformers`.  
2. Load dataset: `from datasets import load\_dataset`.  
3. Retrieve sample Wikipedia articles: `import wikipedia`.  
4. Embed text with sentence-transformers and store in ChromaDB.

## Member 2: LLM & Detection Engineer

Responsibilities:  
- Connect to Gemini Pro using `google-generativeai` SDK.  
- Integrate Gemini with LangChain for modular pipeline.  
- Implement hallucination detection using semantic similarity (threshold tuning).  
- Optionally use Natural Language Inference (NLI) model (`roberta-large-mnli`).  
  
Preferred Technology:  
- Python, LangChain, google-generativeai, Hugging Face Transformers.  
  
Beginner-Friendly Steps:  
1. Install: `pip install google-generativeai langchain transformers`.  
2. Authenticate Gemini: `import google.generativeai as genai`.  
3. Run test query to check output.  
4. Compare answer embeddings against retrieval results for hallucination detection.

## Member 3: Correction & Regeneration Engineer

Responsibilities:  
- Implement self-correction mechanism: regenerate Gemini answer using retrieved evidence.  
- Ensure grounded response is saved alongside original.  
- Add confidence score (similarity measure) and citations to output.  
- Save all results (original, corrected, evidence) into SQLite logs.  
  
Preferred Technology:  
- Python, LangChain, SQLite3, sentence-transformers.  
  
Beginner-Friendly Steps:  
1. Use LangChain `RetrievalQA` to regenerate answer with evidence.  
2. Store outputs in SQLite using Python’s `sqlite3` module.  
3. Create a schema: (Question, RawAnswer, CorrectedAnswer, Citations, ConfidenceScore).  
4. Test correction on small TruthfulQA sample.

## Member 4: Frontend & UX Engineer

Responsibilities:  
- Build CLI interface for offline testing.  
- Develop Streamlit frontend for interactive demo.  
- Connect frontend to Flask backend (LangChain pipeline).  
- Optionally build a VS Code extension (if time allows).  
  
Preferred Technology:  
- Python, Streamlit, Flask, LangChain, (Optional) VS Code API.  
  
Beginner-Friendly Steps:  
1. CLI: use `argparse` in Python for Q&A.  
2. Streamlit: `pip install streamlit`, build text input + output panel.  
3. Flask: define REST routes to call LangChain pipeline.  
4. Integrate Flask API with Streamlit frontend.

## Member 5: Testing & Deployment Engineer

Responsibilities:  
- Benchmark Gemini raw answers vs corrected pipeline on TruthfulQA.  
- Calculate Precision, Recall, F1 using scikit-learn.  
- Visualize results using Matplotlib.  
- Deploy backend (Flask) and frontend (Streamlit Cloud).  
- Prepare final report and presentation.  
  
Preferred Technology:  
- Python, scikit-learn, Matplotlib, Streamlit Cloud, Render/Heroku.  
  
Beginner-Friendly Steps:  
1. Install: `pip install scikit-learn matplotlib`.  
2. Evaluate predictions with `from sklearn.metrics import precision\_recall\_fscore\_support`.  
3. Create bar charts for before/after performance.  
4. Deploy Flask backend on Render, Streamlit app on Streamlit Cloud.