AI-Powered Legal Debate System

1. Approach

The system is designed as an **AI-powered courtroom simulator** where two distinct AI lawyers engage in a structured debate:

- **Prosecution (RAG Lawyer):** Builds **fact-based legal arguments** by retrieving structured information from a small dataset of legal cases (cases.jsonl).
- **Defense (Chaos Lawyer):** Generates **creative and absurd counterarguments**, relying on improvisation rather than evidence.
- **Judge (User + AI Events):** The user plays judge and chooses a winner, while the AI occasionally injects courtroom events (interruptions, warnings) to make the debate lively.

This hybrid approach combines **retrieval-augmented generation (RAG)** with **generative creativity**, ensuring the debate is both informative and entertaining.

2. Retrieval Strategy

The **prosecution lawyer's reasoning** is powered by a **hybrid retrieval strategy** implemented in llm.py:

1. **Corpus Expansion:**
    - The script generate_facts.py transforms each legal case into multiple structured factual snippets (year, jurisdiction, evidence snippets).
    - These facts are stored in facts.py as DEFAULT_FACTS.
2. **Keyword Filtering:**
    - Given a case, keywords are extracted and matched against facts.
    - This ensures that only facts mentioning relevant terms are considered.
3. **Semantic Embedding Search:**
    - The **SentenceTransformer (MiniLM)** model embeds both case text and candidate facts.
    - Cosine similarity is used to rank facts by relevance.
4. **Hybrid Ranking:**
    - Keyword + embedding filtering ensures robustness.
    - Top-k facts (e.g., 5–10) are selected as evidence.
5. **Fact-based Argument Generation:**
    - One fact is selected per round and wrapped in a legal argument template:
      "Based on evidence, [fact]. This strongly supports the prosecution's case."

This ensures that prosecution arguments remain grounded in structured case knowledge, unlike the defense.

. Metadata Structure

The **metadata schema** for each case in cases.jsonl is:


```
{
  "id": 1,
  "title": "Case Title",
  "year": "1998",
  "jurisdiction": "Supreme Court",
  "tags": ["contract", "liability"],
  "text": "Full case description..."
}
```


From this structure, multiple **fact snippets** are generated (e.g., *"Case X was decided in the Supreme Court in 1998"*), which improves retrieval interpretability.

4. System Design

The architecture follows a **modular client–server design**:

- **Backend (FastAPI)**
    - Orchestrates debates (main.py).
    - Loads the corpus (retrieval.py).
    - Generates cases (generator.py).
    - Produces lawyer arguments (llm.py).
    - Summarizes verdicts.
    - Uses **timeout safeguards** so each round completes even if an AI call hangs.
- **Frontend (Streamlit)**
    - Provides interactive UI.
    - Lets users generate cases, start debates, view prosecution vs. defense arguments, and choose verdicts.
    - Displays judge's random interventions and stores debate history.

- **RAG Lawyer vs. Chaos Lawyer**
    - RAG Lawyer = grounded, evidence-based, relies on hybrid retrieval.
    - Chaos Lawyer = imaginative, absurd, generates entertainment value.
- **Judge**
    - Human judge selects winner.
    - AI judge adds random interventions for realism.

5. Challenges Faced

- **No prior knowledge to Streamlit and Fastapi. Having to learn both frontend and backend was probably the biggest issue for me but pretty useful in the future.**
- **Having to use a mock llm due to memory issues and no gpu. Also didn't have access to api keys of Openai, Anthropic,etc.**
- **Though I had basic understanding of an LLM, this was my first real-time application of it and combining it with the lack of web development knowledge made it pretty tough.**