Amlgo Labs: Assignment Tasks

Document Preparation & Ingestion

1. 📁 Folder Chunks – preprocess.py
    a. Import Libraries
    b. Load environment variables from .env file
    c. Define paths and chunks setting
    d. Load PDFs using PyPDFLoader
    e. Clean the text: remove extra whitespace, tabs, and newlines
    f. Sentence-aware text chunking
    g. Save Chunks to JSON file

Output: Chunks.json

2. 📁 Folder Chunks – embed_and_store.py
    a. Import Libraries
    b. Load environment variable from .env file
    c. Define Paths
    d. Load Chunks
    e. Load and create embedding from hugging face
    f. Store in FAISS database

Output: vectordb/index.faiss and index.pkl

IMPROVEMENT

1. Remove HTML tags
2. Remove common header/footer patterns
3. Remove special characters but keep punctuation
4. For more info, we add more things in chunks
5. Instead of JSON, we can save in TXT
6. I Used FAISS, I can use Chroma DB, Qdrant, etc. I used because its fast, Easy to setup.

🧪 Example Use Cases (When to Use)

Use Case	Best Option	Why
🧪 Local development / prototype RAG	**FAISS**	Fast, easy, no setup
🧠 Semantic search over structured docs	**ChromaDB**	Good for metadata filtering, document storage
📦 Scalable backend for retrieval API	**Qdrant**	Production-ready, filtering, deployable via Docker
☁️ Enterprise-grade cloud vector search	**Pinecone**	Fully managed, fast, great for SaaS apps
🔍 Re-ranking + vector + hybrid search	**Qdrant / Pinecone**	Best filtering + hybrid support

7. I Used all-MiniLM-L6-v2 Embedding, we can use bge-small-e and similar. I choose because of fast, lightweight, Good general-purpose and widely adopted.

🧠 Summary Table:

Use Case	Recommended Model	Notes
General-purpose semantic search	`all-MiniLM-L6-v2`	Fast, lightweight
Retrieval optimized (query-doc)	`bge-small-en` / `msmarco-MiniLM`	Add "query:" prefix
Multilingual search	`LaBSE` / `distiluse-multilingual`	Hindi/English mix
Re-ranking (2nd pass)	`bge-reranker-base` / `cross-encoder`	Accurate but slow
Chatbot over PDF/documents (RAG)	`all-MiniLM-L6-v2` or `bge-small`	Start with MiniLM

8. Can use Modular coding with logging & class functionality.
9. Can make Configuration file for chunks and chunk overlap value

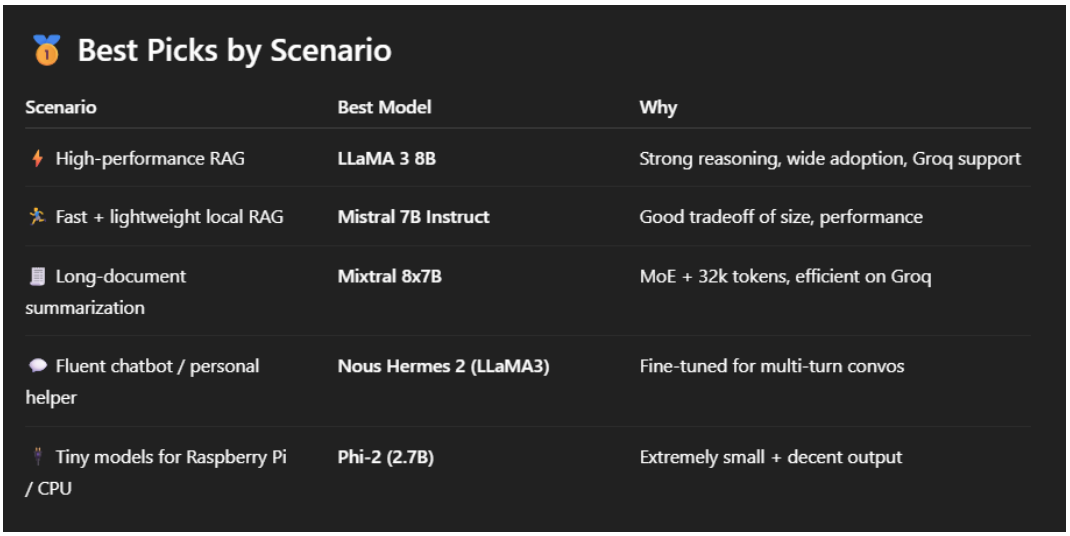
RAG Pipeline

1. 📁 Folder src – retriever.py
    a. Import Libraries
    b. Define DB path
    c. Load Embedding
    d. Function that returns Retriever

2. 📁 Folder src – generator.py
    a. Import Libraries
    b. Load Groq API for LLM
    c. Load LLM
    d. Custom Prompt
    e. Return Langchain prompt Object
3. 📁 Folder src – Pipeline.py
    a. Combine Retriever + Generator

IMPROVEMENT

1. Can use Modular coding with logging & class functionality.
2. Can Apply MLOPS Pipeline using Dag's hub
3. I used llama3 because of Open-source weight model in groq till now, I can use zephyr-7b, mistral-7b.



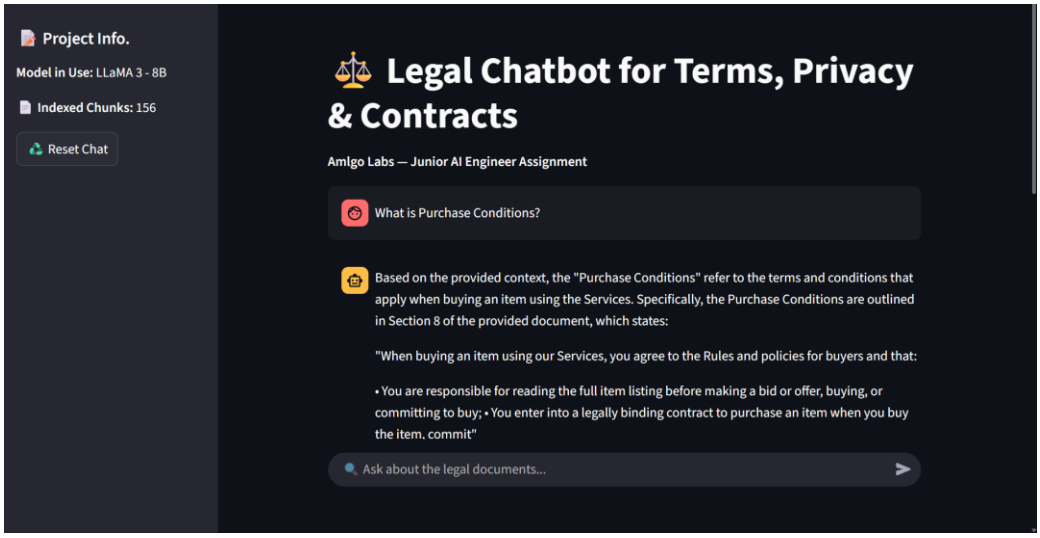
🥇 Best Picks by Scenario

Scenario	Best Model	Why
⚡ High-performance RAG	**LLaMA 3 8B**	Strong reasoning, wide adoption, Groq support
🏃 Fast + lightweight local RAG	**Mistral 7B Instruct**	Good tradeoff of size, performance
📄 Long-document summarization	**Mixtral 8x7B**	MoE + 32k tokens, efficient on Groq
💬 Fluent chatbot / personal helper	**Nous Hermes 2 (LLaMA3)**	Fine-tuned for multi-turn convos
🧠 Tiny models for Raspberry Pi / CPU	**Phi-2 (2.7B)**	Extremely small + decent output

4. Can make Configuration file for K relevant chunks, temperature, Max token & more.
5. Can improve custom prompt template with more strictness in document set (e.g., Terms & Conditions, Privacy Policies, Legal Contracts).

Streamlit UI 💾 App.py

1. Import Libraries
2. Streamlit setup
3. Sidebar Info contain Model name, chunks, reset button
4. Initialize chat history
5. Display past message
6. Chat



IMPROVEMENT

1. Add more details on sidebar about model and embedding
2. Functionality to select LLM and Embedding model
3. Functionality to select chunks
4. Deploy….

Description of document structure and chunking logic:

Document Structure

The system processes legal documents such as:

- Terms & Conditions
- Privacy Policies
- Legal Contracts

These documents are typically long, formal, and sectioned into clauses or numbered headings. Each document is loaded and processed as a collection of text pages, typically via PDF.

The pipeline loads documents using:

from langchain_community.document_loaders import PyPDFLoader

Chunking Logic

To enable accurate retrieval and semantic search, documents are split into smaller "chunks". These chunks are used to build vector embeddings for similarity search.

Here's the chunking strategy used:

Parameter	Value
Chunk Size	~300 words per chunk
Overlap	50 words between chunks
Split Method	Sentence-aware splitting
Tool Used	nltk.sent_tokenize()

Why Sentence-Aware Chunking?

Legal documents require high factual precision. Instead of splitting blindly by character length, we use sentence-aware splitting.

Example Chunk Output:

Chunk 1:

"The user agrees to abide by all terms stated in this agreement. This includes usage limits and data privacy clauses as described herein."

Chunk 2 (with overlap):

"This includes usage limits and data privacy clauses as described herein. The company reserves the right to terminate access at any time..."

Overlap ensures context continuity, improving LLM understanding and accuracy during retrieval.

Tech Summary:

- Documents loaded using PyPDFLoader
- Cleaned to remove \n, tabs, etc.
- Chunked using nltk-based sentence splitter
- Chunks wrapped as langchain.Document objects with optional metadata (source, section)
- Stored in FAISS for fast semantic retrieval

Explanation of embedding model and vector DB used

Embedding Model: all-MiniLM-L6-v2

Model Info:

- Name: sentence-transformers/all-MiniLM-L6-v2
- Source: HuggingFace Transformers
- Architecture: Based on MiniLM (miniaturized BERT)
- Vector Size: 384 dimensions

Why It Was Used:

- Lightweight & Fast: Ideal for real-time applications like chatbots.
- Semantic Understanding: Good at capturing sentence-level meaning, which is essential for retrieving relevant chunks.
- Compatible: Fully supported by LangChain & FAISS, integrates easily with HuggingFace Embeddings interface.

How It's Used:

Each document chunk is converted into a dense vector (embedding) that represents its semantic meaning. These vectors are stored and later searched based on similarity to the user's query vector.

```
from langchain_huggingface import HuggingFaceEmbeddings

embedding_model = HuggingFaceEmbeddings(

    model_name="sentence-transformers/all-MiniLM-L6-v2"

)
```

Vector Store: FAISS (Facebook AI Similarity Search)

Tech Overview:

- Developed by Meta AI (Facebook)
- Used for fast nearest neighbor search on high-dimensional vectors. Highly optimized in C++, with Python bindings.

Why FAISS Was Used:

- High-Speed Retrieval: In-memory vector search is extremely fast.
- Persistent Storage: Vectors can be saved/loaded via disk.
- LangChain Native Support: Simple plug-and-play integration with retrievers.

How It's Used:

All embedding vectors created from document chunks are stored in FAISS. When a user submits a query, its embedding is compared to the stored ones, and top-k (e.g., 3) most similar chunks are retrieved.

```
from langchain_community.vectorstores import FAISS
```

db = FAISS.from_documents(chunks, embedding_model)

db.save_local("vectorstore/db_faiss")

During QA:

retriever = db.as_retriever(search_kwargs={"k": 3})

I used the all-MiniLM-L6-v2 embedding model due to its excellent balance between performance and speed, making it ideal for real-time legal QA. FAISS was chosen as the vector store for its high-speed retrieval, in-memory search capabilities, and native LangChain support allowing seamless integration and efficient semantic search over legal document chunks.

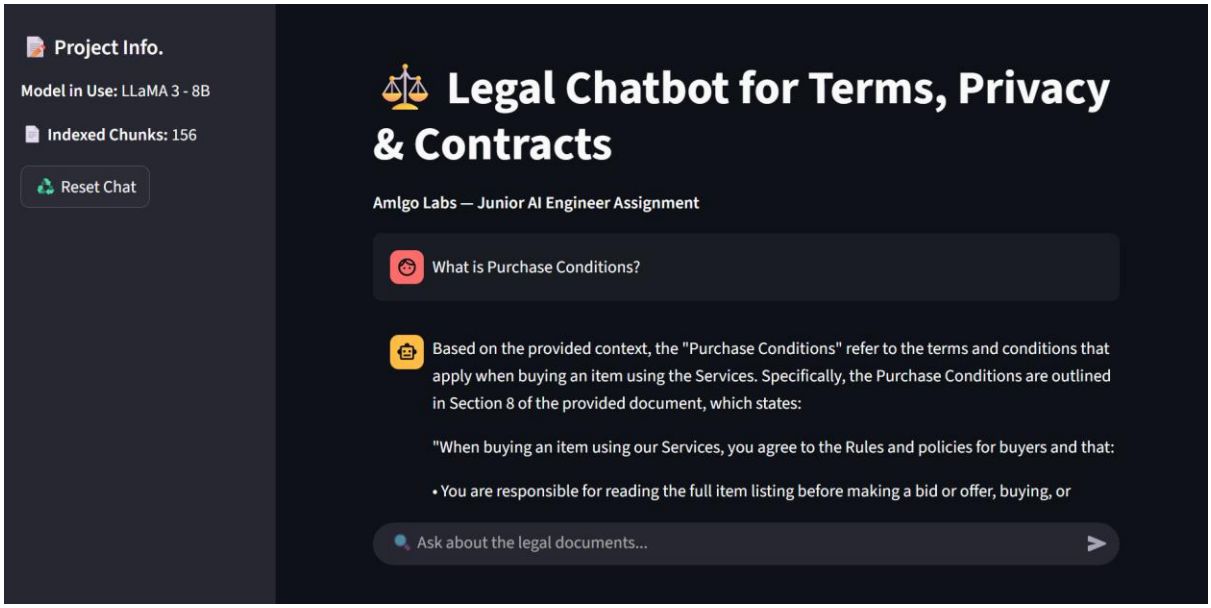
Prompt format and generation logic

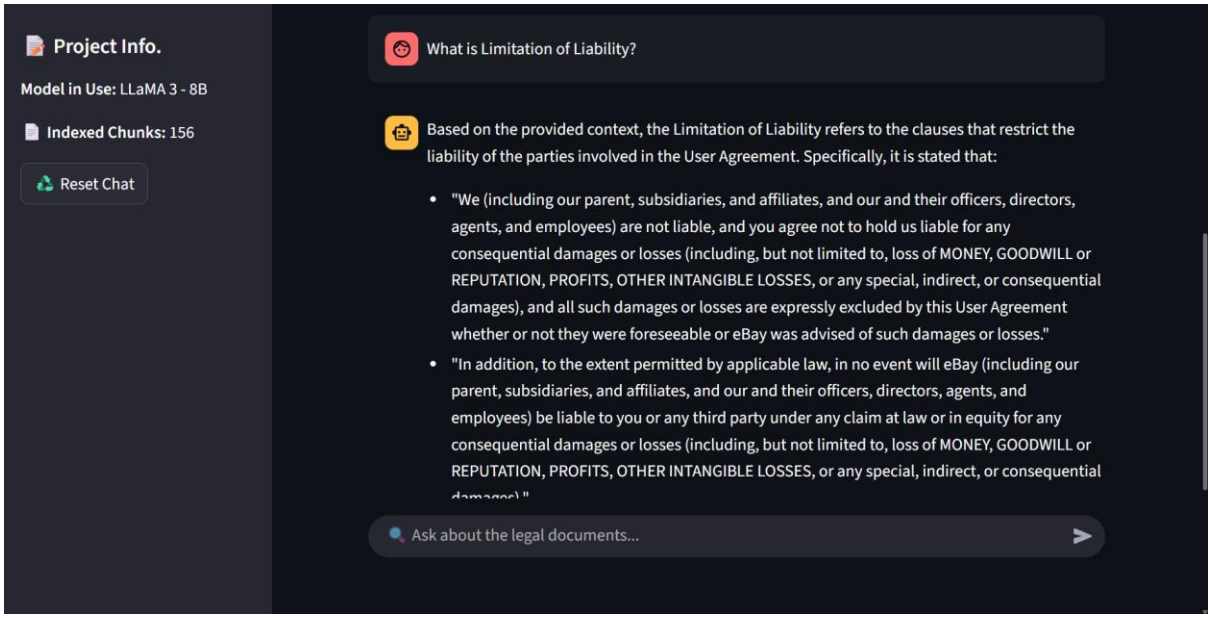
Role Assignment:

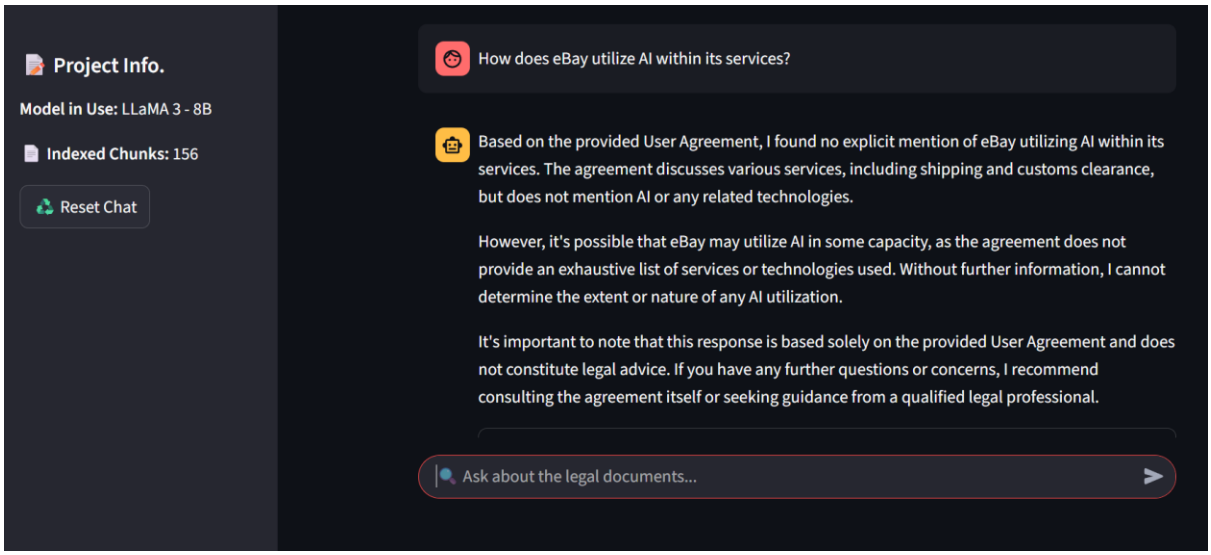
- Sets clear model persona: an assistant for legal documents.
- Improves tone, formatting, and focus of responses.
- Prevents hallucination
- Encourages interpretability of legal terms
- Adds credibility by suggesting citations
- Maintains safety ("do not give legal advice")

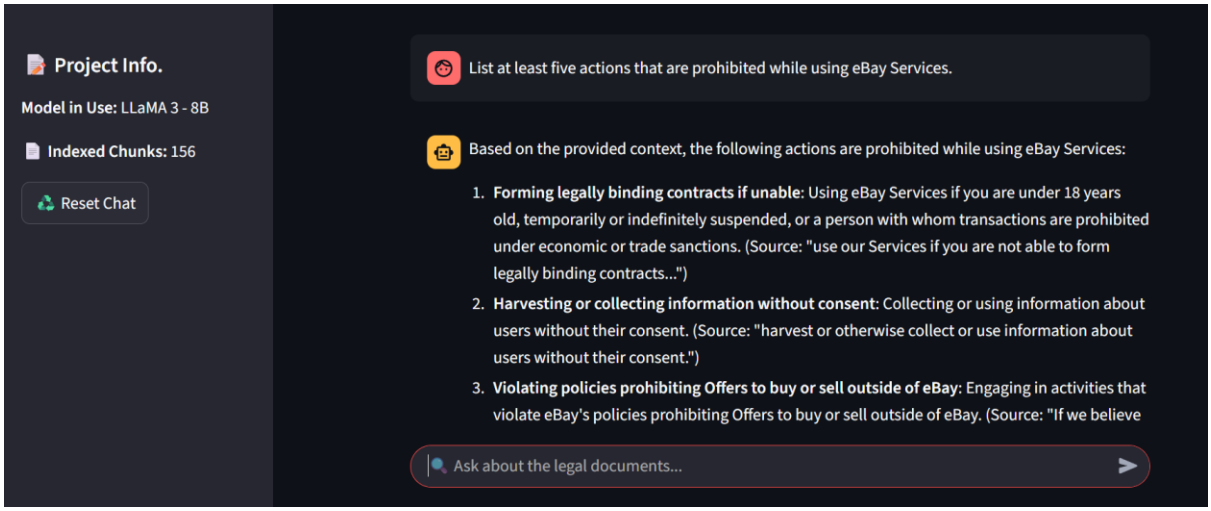
I created a structured prompt that acts like a legal document assistant. It ensures factual, transparent answers by enforcing five key instructions: source-only answers, no assumptions, clarity, citations, and honest fallback when info is missing.
The format is segmented clearly (context, question, guidelines, response), which improves accuracy and reduces hallucination especially critical for legal documents.

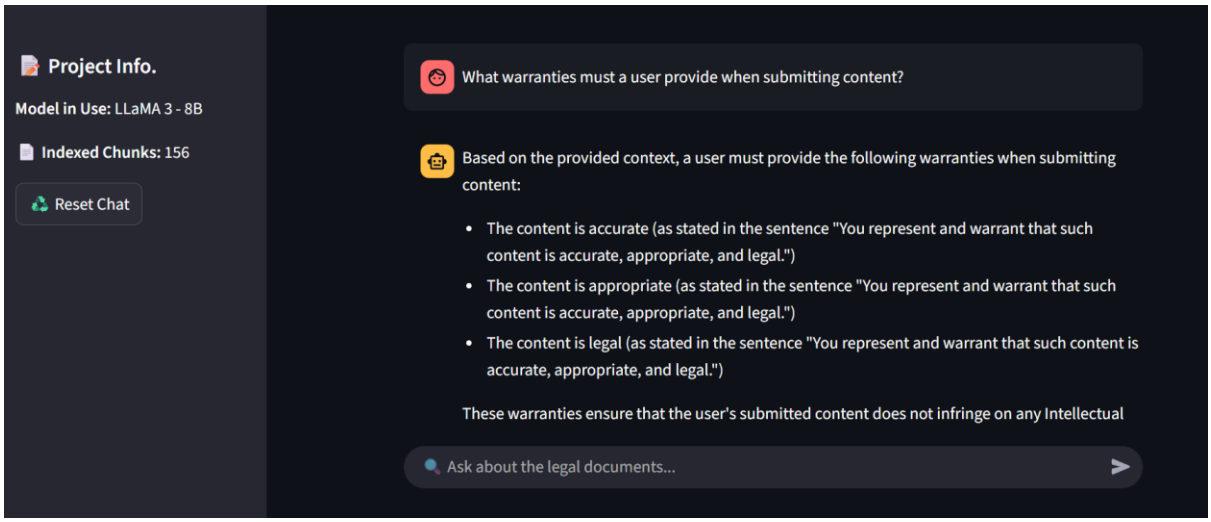
At least 3–5 example queries with responses (highlight success and failure cases)


Project Info.
Model in Use: LLaMA 3 - 8B
Indexed Chunks: 156
Reset Chat
⚖️ Legal Chatbot for Terms, Privacy & Contracts
Amlgo Labs — Junior AI Engineer Assignment
What is Purchase Conditions?
Based on the provided context, the "Purchase Conditions" refer to the terms and conditions that apply when buying an item using the Services. Specifically, the Purchase Conditions are outlined in Section 8 of the provided document, which states:
"When buying an item using our Services, you agree to the Rules and policies for buyers and that:
• You are responsible for reading the full item listing before making a bid or offer, buying, or
Ask about the legal documents...


Project Info.
Model in Use: LLaMA 3 - 8B
Indexed Chunks: 156
Reset Chat
What is Limitation of Liability?
Based on the provided context, the Limitation of Liability refers to the clauses that restrict the liability of the parties involved in the User Agreement. Specifically, it is stated that:
"We (including our parent, subsidiaries, and affiliates, and our and their officers, directors, agents, and employees) are not liable, and you agree not to hold us liable for any consequential damages or losses (including, but not limited to, loss of MONEY, GOODWILL or REPUTATION, PROFITS, OTHER INTANGIBLE LOSSES, or any special, indirect, or consequential damages), and all such damages or losses are expressly excluded by this User Agreement whether or not they were foreseeable or eBay was advised of such damages or losses."
"In addition, to the extent permitted by applicable law, in no event will eBay (including our parent, subsidiaries, and affiliates, and our and their officers, directors, agents, and employees) be liable to you or any third party under any claim at law or in equity for any consequential damages or losses (including, but not limited to, loss of MONEY, GOODWILL or REPUTATION, PROFITS, OTHER INTANGIBLE LOSSES, or any special, indirect, or consequential
Ask about the legal documents...


Project Info.
Model in Use: LLaMA 3 - 8B
Indexed Chunks: 156
Reset Chat
How does eBay utilize AI within its services?
Based on the provided User Agreement, I found no explicit mention of eBay utilizing AI within its services. The agreement discusses various services, including shipping and customs clearance, but does not mention AI or any related technologies.
However, it's possible that eBay may utilize AI in some capacity, as the agreement does not provide an exhaustive list of services or technologies used. Without further information, I cannot determine the extent or nature of any AI utilization.
It's important to note that this response is based solely on the provided User Agreement and does not constitute legal advice. If you have any further questions or concerns, I recommend consulting the agreement itself or seeking guidance from a qualified legal professional.
Ask about the legal documents...





Notes on hallucinations, model limitations, or slow responses

The chatbot is designed to minimize hallucinations by enforcing strict source-only prompts and fallback responses.
It avoids legal misinterpretation by instructing the LLM to explain document language, not give legal advice.
While LLaMA 3 is a large model, using Groq ensures low-latency streaming responses under 2 seconds for most queries.