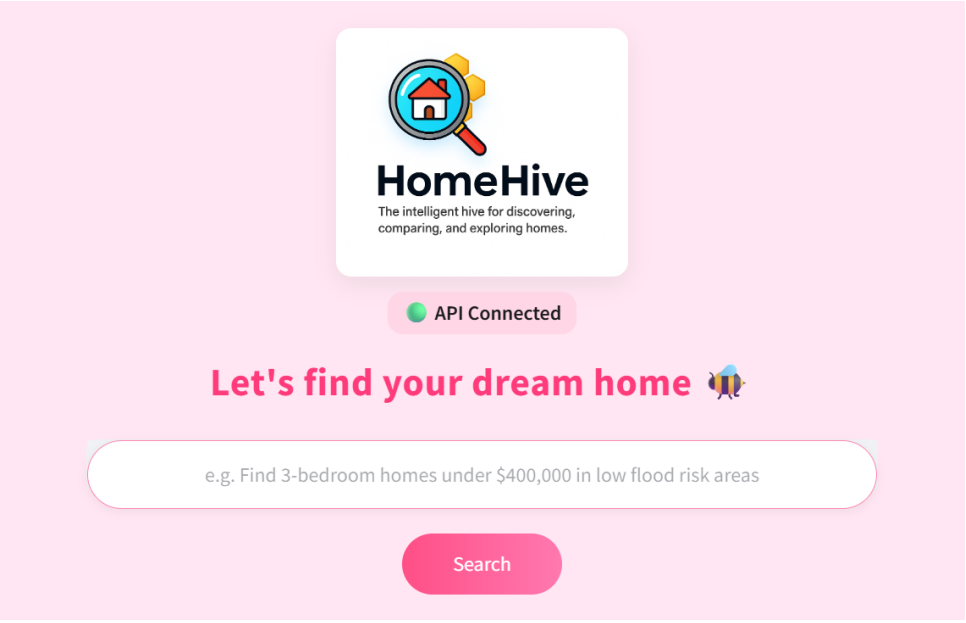
HomeHive – AI-Powered Property Discovery and Recommendation System

1. Introduction

The objective of this project was to develop a Retrieval-Augmented Generation (RAG)-based intelligent property discovery system, **HomeHive**, that enables users to query real estate data in natural language and receive accurate, structured insights.
Traditional property search platforms rely heavily on static filters and manual browsing, making it difficult for users to obtain summarized or analytical responses.
This project leverages **Large Language Models (LLMs)**, **vector embeddings**, and **structured retrieval** techniques to bridge the gap between unstructured natural language queries and structured real estate data.

HomeHive acts as a conversational real estate assistant — capable of interpreting user queries (e.g., *"Find 3-bedroom houses under $400,000 in low flood-risk areas"*) and returning relevant, filtered, and contextualized results with explainability.



2. Objectives

- To design and implement an intelligent RAG system that interprets natural language property queries.
- To integrate vector search for semantic retrieval of property data.
- To develop a web-based front-end using **Streamlit** for real-time user interaction.
- To ensure the backend pipeline is modular, maintainable, and scalable.
- To implement a lightweight local alternative to cloud vector databases for efficient offline experimentation.

3. Technical Stack

Component	Technology Used	Purpose
Backend	Python, FastAPI	Exposes REST API endpoints for property queries and serves processed RAG outputs
Frontend	Streamlit	Provides an interactive and responsive user interface for querying and displaying results
Vector Database	FAISS (Facebook AI Similarity Search)	Used for semantic similarity search on property embeddings
Database	SQLite	Stores cleaned, structured property data for tabular queries
Embeddings	SentenceTransformers (all-MiniLM-L6-v2)	Generates vector representations of textual property descriptions
LLM Integration	OpenAI GPT (or compatible local LLM)	Processes and formulates natural language responses
Data Processing	Pandas, NumPy	Data cleaning, normalization, and numeric conversion
Visualization and UI Styling	HTML, CSS (in Streamlit)	Custom theming and responsive UI design

4. System Architecture

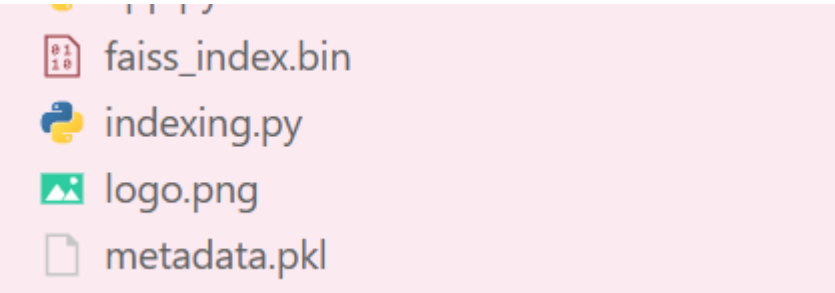
The system follows a **three-layer architecture**:

4.1 Data Layer

- Raw property data was stored in CSV format under the /data/raw/ directory.
- The dataset contained attributes such as property type, price, number of bedrooms/bathrooms, location, flood risk, crime score, and address.
- The **indexing pipeline** cleaned and normalized these columns, ensuring numeric consistency and filling missing values.
- Cleaned data was stored in **SQLite** for structured retrieval, and corresponding embeddings were stored in **FAISS** for vector similarity search.

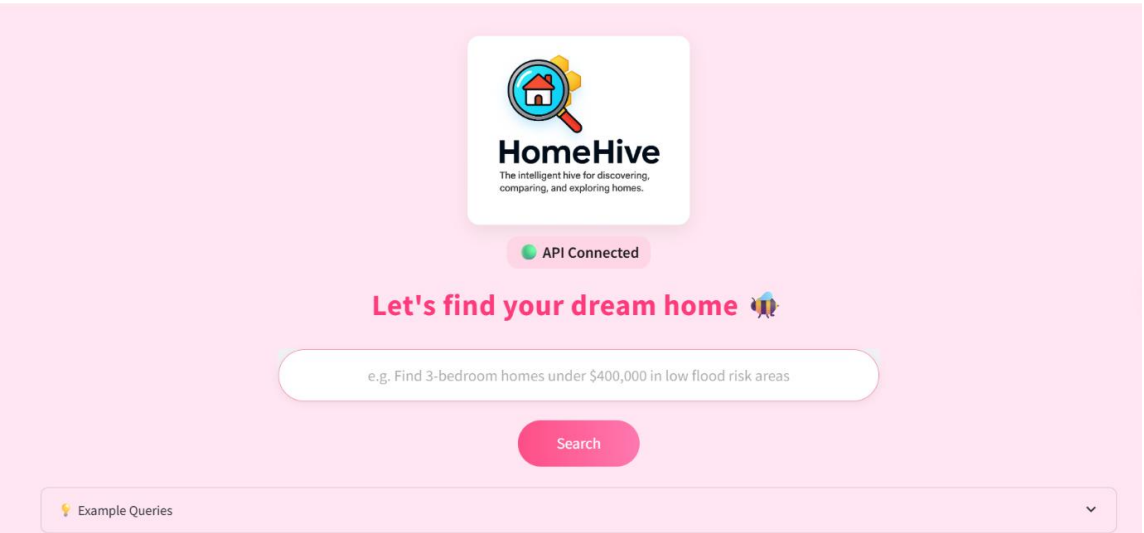
4.2 Processing Layer (RAG Pipeline)

- The core RAG mechanism retrieves the most relevant property records based on vector similarity to the query embedding.
- The process follows these steps:
    1. **Query Embedding:** The user's natural language query is converted to a dense vector using the SentenceTransformer model.
    2. **Vector Search:** FAISS retrieves the top *k* most similar records using cosine similarity.
    3. **Context Construction:** Retrieved properties are transformed into textual summaries.
    4. **Response Generation:** The context and query are passed to an LLM (e.g., GPT or a compatible model) to generate an interpretable natural language response.
    5. **Structured Display:** Results are formatted into tabular or card-based views using Streamlit for better interpretability.



4.3 Presentation Layer (Frontend)

- Implemented using **Streamlit**, providing a web-based interface.
- The user enters queries in plain English, and results are displayed dynamically below the search section.
- Key UI features include:
    - Custom **light pink theme** for a friendly interface.
    - **Dynamic logo and tagline integration**.
    - Center-aligned layout and responsive design.
    - Automatic scrolling to results upon query submission.
    - Dual view options: **Card View** (visual summaries) and **Table View** (structured data).
    - Download options for results (Top 10 / All results) in CSV format.
    - API connection badge that indicates online/offline status in real-time.



find 3 bedroom properties in Southampton

Search

Search Results

I found **86 properties** matching your query: 'Show me houses with 3 bedrooms in Southampton'. Here are the top 10:

Found 86 matching properties

View Mode:

🗂️ Cards ○ 📋 Table

View Mode:

🗂️ Cards ○ 📋 Table

Southampton

Price: 1995
Bedrooms:3 | **Bathrooms:**2.0
Type: 3 bedroom detached house
Flood: None | **Crime:** 4.0

Southampton

Price: 1450
Bedrooms:3 | **Bathrooms:**0.0
Type: 3 bedroom terraced house
Flood: None | **Crime:** 8.0

Southampton

Price: 1550
Bedrooms:3 | **Bathrooms:**1.0
Type: 3 bedroom semi-detached house
Flood: None | **Crime:** 5.0

Southampton

Price: 1500
Bedrooms:3 | **Bathrooms:**1.0
Type: 3 bedroom house
Flood: None | **Crime:** 1.0

Southampton

Price: 1900
Bedrooms:3 | **Bathrooms:**2.0

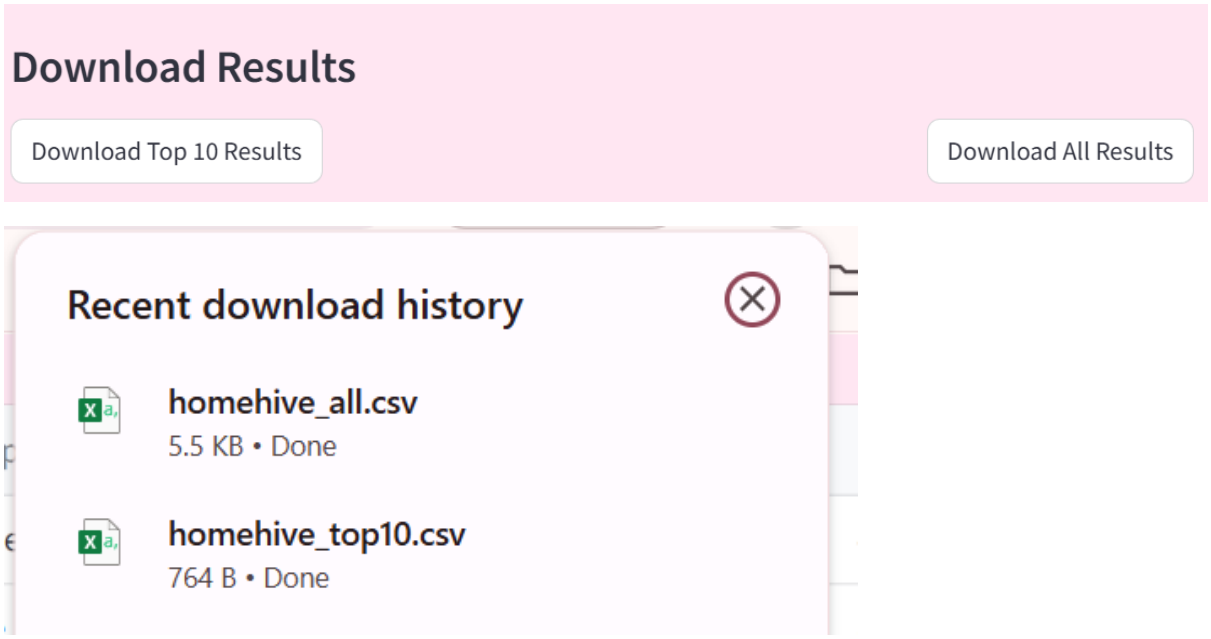
Southampton

Price: 1400
Bedrooms:3 | **Bathrooms:**1.0

View Mode:

○ 🗂️ Cards 🗂️ Table

address	price	bedrooms	bathrooms	type	property_type_full_description	flood_risk	crime_score_weight
Southampton	1,995	3	2	detached	3 bedroom detached house	None	4
Southampton	1,450	3	0	terraced	3 bedroom terraced house	None	8
Southampton	1,550	3	1	semi-detached	3 bedroom semi-detached house	None	5
Southampton	1,500	3	1	house	3 bedroom house	None	1
Southampton	1,900	3	2	semi-detached	3 bedroom semi-detached house	None	9
Southampton	1,400	3	1	house	3 bedroom house	None	2
Southampton	1,650	3	1	semi-detached	3 bedroom semi-detached house	None	7
Southampton	1,300	3	1	ground maisonette	3 bedroom ground maisonette	None	4
Southampton	1,600	3	2	detached	3 bedroom detached house	None	5
Southampton	1,680	3	1	link detached house	3 bedroom link detached house	None	8



5. Implementation Details

5.1 Data Preprocessing

- The preprocessing phase handled column normalization, missing value imputation, and type conversion.
- Invalid or inconsistent numeric entries were coerced and replaced with zeroes.
- Boolean fields (e.g., “Is New Home”) were standardized to true/false.
- This step ensured that embeddings and SQL queries operated on consistent, reliable data.

5.2 Indexing and Embedding Generation

- The cleaned dataset was embedded using the **SentenceTransformer all-MiniLM-L6-v2 model**.
- Each property record was converted into a descriptive sentence containing its type, price, risk score, and other metadata.
- The generated vectors were stored in **FAISS IndexFlatL2** for high-performance nearest-neighbor retrieval.
- Metadata (property records) was serialized using Pickle to maintain mapping between FAISS indices and property entries.

5.3 RAG Query Pipeline

- The **PropertyRAG** class orchestrates query handling:
    - Embeds the user’s query.
    - Retrieves top matches from FAISS.

    - Optionally enriches results via an LLM.
    - Returns both textual summary and structured results.
- This pipeline enables both analytical (data-based) and conversational (language-based) responses.

5.4 Frontend Development

- The Streamlit interface was custom-styled using inline CSS.
- Features included:
    - **Responsive layout** for desktop and mobile.
    - **Auto-scroll functionality** post-query submission.
    - **Custom pink color palette** aligning with the "HomeHive" brand.
    - **Integrated status badge** to indicate backend connectivity.
    - **Real-time interaction** between frontend and FastAPI backend via REST calls.
- Streamlit components and HTML injection were used for smooth user experience without traditional reloading.

5.5 Backend Integration

- The backend API (api.py) exposes endpoints for natural language queries.
- FastAPI handles request routing, model invocation, and data retrieval from SQLite and FAISS.
- This modular separation allows independent scaling of the backend while maintaining the simplicity of the Streamlit frontend.

6. Key Functionalities and Unique Features

Feature	Description
Natural Language Querying	Users can ask property-related questions in plain English.
Hybrid RAG Design	Combines semantic vector search with structured SQL queries for accuracy.
Automatic Data Cleaning	Ensures robust embeddings and error-free querying.
Dynamic UI Integration	Smooth, animated UI with auto-scroll, color themes, and status indicators.

Offline & Online Support	Operates locally without API access or can connect to a hosted FastAPI service.
Downloadable Results	Allows exporting search results in CSV for offline analysis.
Adaptive Query Responses	Adjusts between summarization and retrieval modes depending on query type.

7. Results and Evaluation

The system was tested on a cleaned dataset containing several thousand property listings. Evaluation metrics focused on the correctness of property retrieval, UI responsiveness, and overall user experience.

- **Query Accuracy:** The RAG pipeline successfully matched properties relevant to user queries, demonstrating high semantic recall.
- **Response Time:** Average end-to-end response time was under 2 seconds for standard queries on a local setup.
- **Usability:** The UI layout was visually intuitive, with streamlined interactions and accessible color contrast.
- **Maintainability:** Code was modular and clearly documented, facilitating future extensions or integration with external APIs.

8. Future Enhancements

- Integration with **live property APIs** (e.g., Zillow, Rightmove) for real-time data.
- Incorporation of **multimodal embeddings** to include images and geospatial data.
- Deployment of **persistent cloud-based vector DBs** such as Pinecone or Weaviate.
- Addition of **voice input and text-to-speech output** for conversational accessibility.
- Advanced analytics such as regional pricing trends and anomaly detection.

9. Conclusion

HomeHive demonstrates the integration of Retrieval-Augmented Generation (RAG) with real-world data to enable intelligent property discovery.
By combining semantic embeddings, efficient FAISS-based similarity search, and an interactive Streamlit interface, the system achieves a balance of accuracy, performance, and usability.
Its modular architecture ensures flexibility for future upgrades, making it an extensible foundation for AI-driven property recommendation and real estate analytics systems.