Key Point: Early warning system

Functionality: Identifies unusual patterns in sensor or usage data to flag

potential issues.

**Multimodal Input Support**

**Key Point**: Flexible data handling

**Functionality:** Accepts text, PDFs, and CSVs for document analysis and

forecasting.

**Streamlit or Gradio UI**

**Key Point:** User-friendly interface

**Functionality:** Provides an intuitive dashboard for both citizens and city officials

to interact with the assistant.

**3. Architecture**

**Frontend (Stream lit):**

The frontend is built with Stream lit, offering an interactive web UI with

multiple pages including dashboards, file uploads, chat interface, feedback

forms, and report viewers. Navigation is handled through a sidebar using the

stream lit-option-menu library. Each page is modularized for scalability.

**Backend (Fast API):**

Fast API serves as the backend REST framework that powers API endpoints for

document processing, chat interactions, eco tip generation, report creation,

and vector embedding. It is optimized for asynchronous performance and easy

Swagger integration.

**LLM Integration (IBM Watsonx Granite):**

Granite LLM models from IBM Watsonx are used for natural language

understanding and generation. Prompts are carefully designed to generate

summaries, sustainability tips, and reports.

**Vector Search (Pinecone):**

Uploaded policy documents are embedded using Sentence Transformers and

stored in Pinecone. Semantic search is implemented using cosine similarity to

allow users to search documents using natural language queries.

ML Modules (Forecasting and Anomaly Detection):

Lightweight ML models are used for forecasting and anomaly detection using

Scikit-learn. Time-series data is parsed, modeled, and visualized using pandas

and matplotlib.

**4. Setup Instructions**

**Prerequisites:**

* Python 3.9 or later
* pip and virtual environment tools
* API keys for IBM Watsonx and Pinecone
* Internet access to access cloud services
* Installation Process:
  + - * Clone the repository
      * Install dependencies from requirements.txt
      * Create a .env file and configure credentials
      * Run the backend server using Fast API
      * Launch the frontend via Stream lit
      * Upload data and interact with the modules

**5. Folder Structure**

app/ – Contains all Fast API backend logic including routers, models, and

integration modules.

app/api/ – Subdirectory for modular API routes like chat, feedback, report, and

document vectorization.

ui/ – Contains frontend components for Stream lit pages, card layouts, and

form UIs.

smart\_dashboard.py – Entry script for launching the main Stream lit

dashboard.

granite\_llm.py – Handles all communication with IBM Watsonx Granite model

including summarization and chat.

document\_embedder.py – Converts documents to embeddings and stores in

Pinecone.

kpi\_file\_forecaster.py – Forecasts future energy/water trends using regression.

anomaly\_file\_checker.py – Flags unusual values in uploaded KPI data.

report\_generator.py – Constructs AI-generated sustainability reports.

**6. Running the Application**

**To start the project:**

* ➢ Launch the FastAPI server to expose backend endpoints.
* ➢ Run the Streamlit dashboard to access the web interface.
* ➢ Navigate through pages via the sidebar.
* ➢ Upload documents or CSVs, interact with the chat assistant, and view
* outputs like reports, summaries, and predictions.
* ➢ All interactions are real-time and use backend APIs to dynamically
* update the frontend.

**Frontend (Stream lit):**

The frontend is built with Stream lit, offering an interactive web UI with

multiple pages including dashboards, file uploads, chat interface, feedback

forms, and report viewers. Navigation is handled through a sidebar using the

stream lit-option-menu library. Each page is modularized for scalability.

**Backend (Fast API):**

Fast API serves as the backend REST framework that powers API endpoints for

document processing, chat interactions, eco tip generation, report creation,

and vector embedding. It is optimized for asynchronous performance and easy

Swagger integration.

**7. API Documentation**

**Backend APIs available include:**

**POST /chat/ask** – Accepts a user query and responds with an AI-generated

message

**POST /upload-doc** – Uploads and embeds documents in Pinecone

**GET /search-docs** – Returns semantically similar policies to the input query

**GET /get-eco-tips** – Provides sustainability tips for selected topics like energy,

water, or waste

**POST /submit-feedback** – Stores citizen feedback for later review or analytics

Each endpoint is tested and documented in Swagger UI for quick inspection

and trial during development.

**8. Authentication**

* each endpoint is tested and documented in Swagger UI for quick inspection

and trial during development.

* This version of the project runs in an open environment for demonstration.

**However, secure deployments can integrate:**

* Token-based authentication (JWT or API keys)
* OAuth2 with IBM Cloud credentials
* Role-based access (admin, citizen, researcher)
* Planned enhancements include user sessions and history tracking.8.
* Authentication

**9. User Interface**

* The interface is minimalist and functional, focusing on accessibility for nontechnical users. It includes:
* Sidebar with navigation
* KPI visualizations with summary cards
* Tabbed layouts for chat, eco tips, and forecasting
* Real-time form handling
* PDF report download capability
* The design prioritizes clarity, speed, and user guidance with help texts and
* intuitive flows.

**10. Testing**

Testing was done in multiple phases:

**Unit Testing**: For prompt engineering functions and utility scripts

**API Testing**: Via Swagger UI, Postman, and test scripts

**Manual Testing:** For file uploads, chat responses, and output consistency

**Edge Case Handling:** Malformed inputs, large files, invalid API keys

Each function was validated to ensure reliability in both offline and APIconnected modes.