Job Email Assistant

Team Members:
Siddharth Saravanan
Aswin Balaji Thippa Ramesh
Rahul Aravind

Problem Statement

Addressing the Challenges of Job Application Management

Job seekers today face **overwhelming challenges** in managing their application-related emails. With numerous platforms sending confirmations, updates, and invites, it becomes difficult to keep track of important communications. This often leads to missed opportunities, confusion, and unorganized inboxes, hindering the job search process.

Key Challenges

Email Overload

Important messages can get lost easily

Lack of Clarity

Job application statuses remain unclear and confusing

No Structure

Tracking progress requires a lot of effort

Objectives

Structured Records

Convert emails into actionable job records

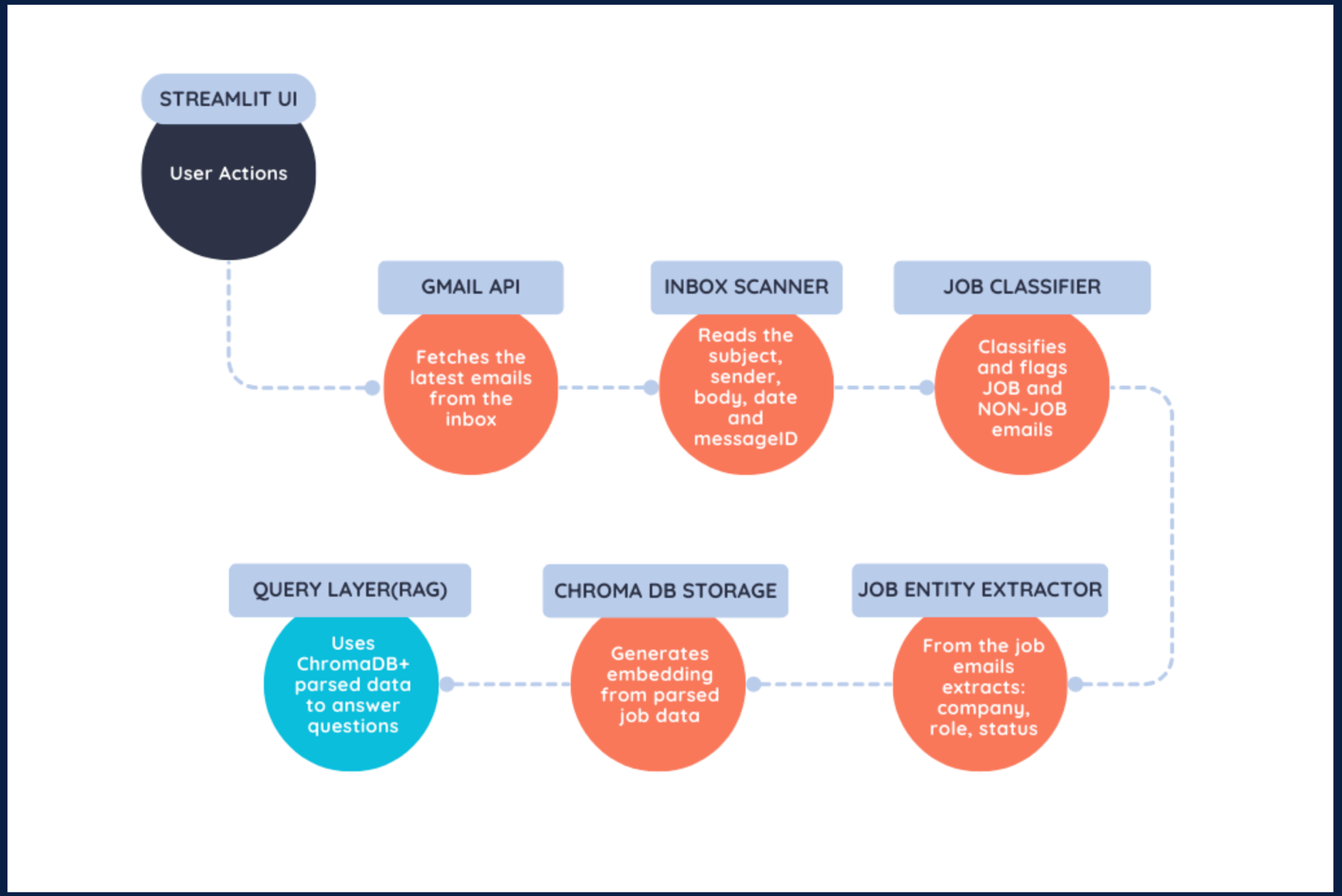
Status Tracking

Keep track of application statuses efficiently

Interaction

Agent that helps interacting with the job applications

Workflow


STREAMLIT UI
User Actions
GMAIL API
Fetches the latest emails from the inbox
INBOX SCANNER
Reads the subject, sender, body, date and messageID
JOB CLASSIFIER
Classifies and flags JOB and NON-JOB emails
JOB ENTITY EXTRACTOR
From the job emails extracts: company, role, status
CHROMA DB STORAGE
Generates embedding from parsed job data
QUERY LAYER(RAG)
Uses ChromaDB+ parsed data to answer questions

Gmail Extraction

1. Authentication & Query

- Securely connect to Gmail via OAuth 2.0
- Use it to filter the emails
- Fetch all matching Message IDs.

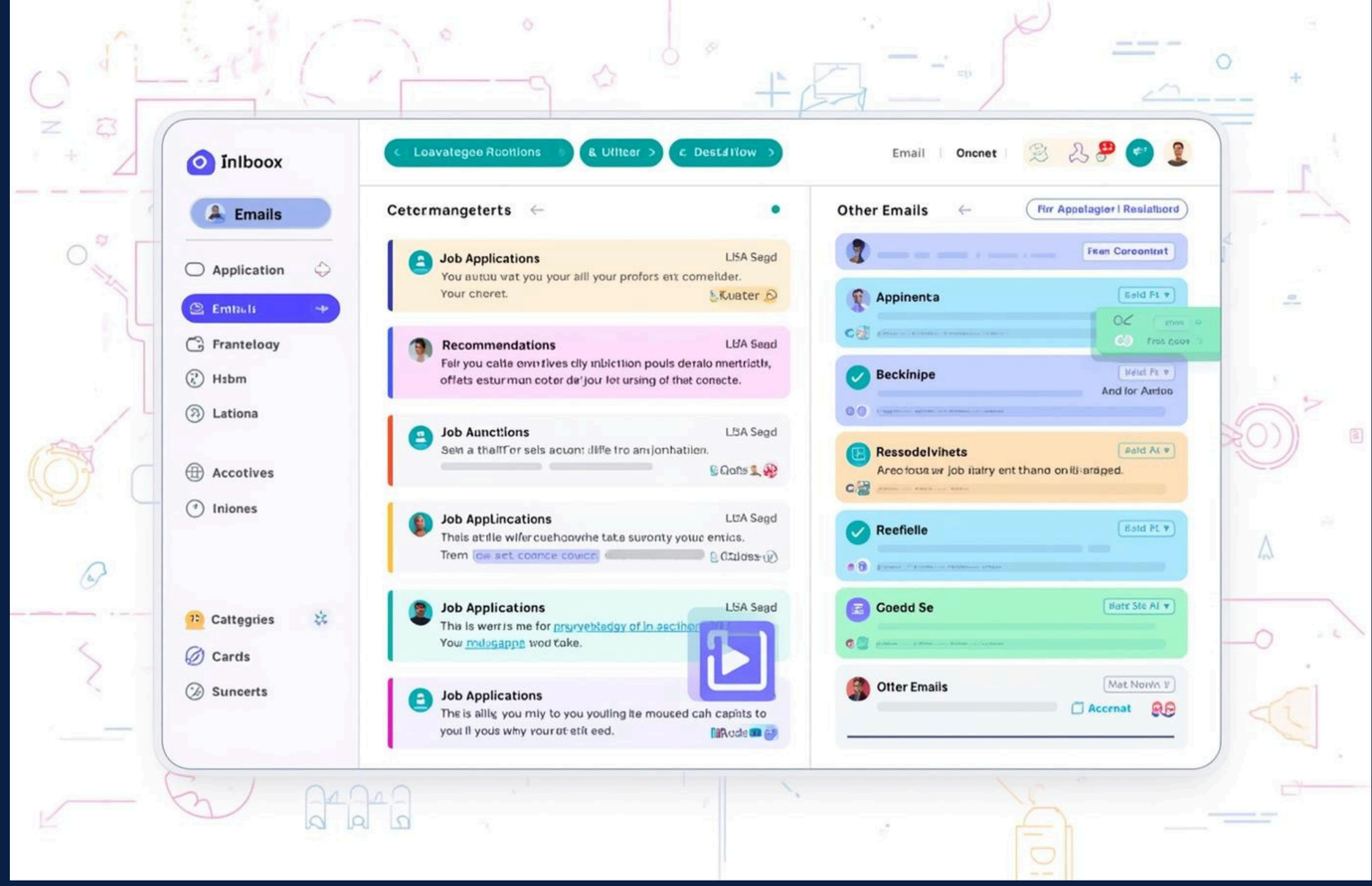
2. Parsing and Cleaning

- Fetch the raw email data for each ID.
- Use BeautifulSoup to convert messy HTML email bodies into clean, reliable plain text.

3. Structure & Export

- Collect all extracted fields: Sender, Subject, Body, Date and a direct Gmail Link.
- Build a Pandas DataFrame from the collected data.
- Export the complete table to a final, analysis-ready Excel file (`.xlsx`).

Inbox Email Classifier: Classifying Job Application Emails



Email Classification

Classifies and organizes emails into structured categories efficiently.

Classifier I

Implemented a classic text classification pipeline to distinguish between "job" and "non_job" related emails.
The process starts by loading and balancing the imbalanced dataset via downsampling. Email subject and body are merged into a single text feature.
The core is an Scikit-learn Pipeline combining two steps:

- TF-IDF Vectorization: Converts text into numerical features, giving higher weight to important, less frequent words.
- Logistic Regression: A highly effective, linear classifier trained to predict the email's label.

The model is trained, evaluated using a classification report, and finally serialized using joblib for deployment, saving the entire workflow as job_classifier_baseline.pkl.

Classifier II

The process starts by combining the email Subject + Body into a single text feature. The text is normalized (lowercased, cleaned) and ready for the multi-layered classification logic.

Rule-Based High Precision: Apply filters for definitive outcomes:

- IF matches job-process (e.g., "Interview invitation," "Job offer") → JOB.
- ELSE IF matches job-alert (e.g., "Items," "Product") → NON-JOB.

Semantic Scoring (Ambiguous): For remaining emails, use SentenceTransformer embeddings:

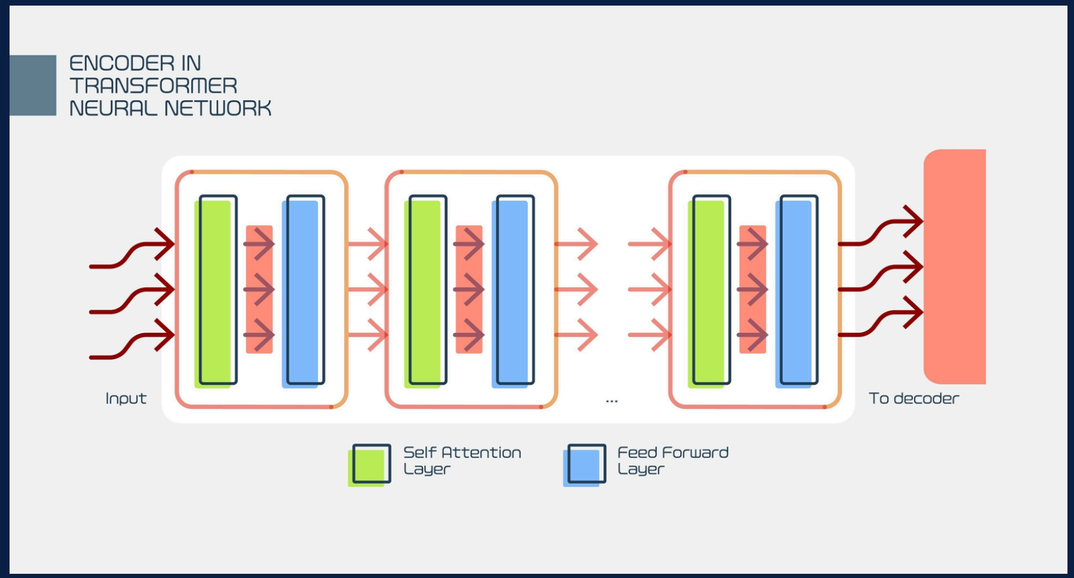
- Calculate Cosine Similarity to known job examples.
- IF Similarity ›= 0.82 AND core keywords are present, THEN → JOB.

Final Default: All remaining emails → NON-JOB.

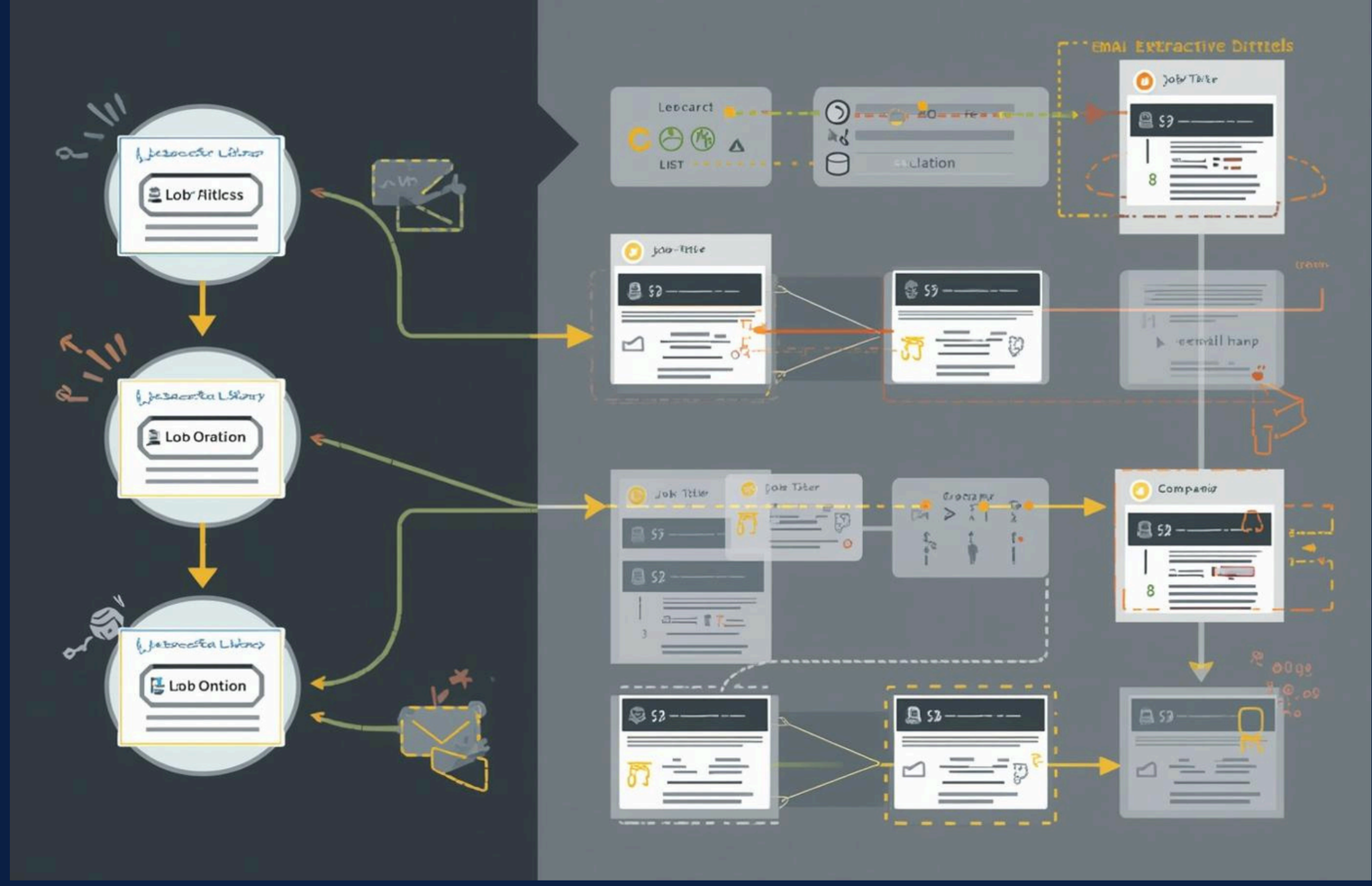
Classifier III

The process begins by balancing the dataset and tokenizing the combined email text and subject fields for the model. The data is then loaded into the HuggingFace Dataset format.

We fine-tune the pre-trained DistilBert-base-uncased model using the Trainer API over three epochs. Finally, the fine-tuned model and tokenizer are saved directly to a folder, and a joblib wrapper is created for simple, one-line loading and prediction in deployment, making the powerful BERT architecture accessible for inference.



Named Entity Recognition Agent in Job Parsing



Structured Data Extraction

The NER Agent utilizes regex and spaCy for extraction.

Logic and Functionality

This script performs Named Entity Recognition (NER) by leveraging a local Large Language Model (LLM), Ollama running Llama 3.1, to extract structured data from job emails.

The LLM is tasked with extracting four specific entities: "company_name", "position_applied", "application_date", and "status". To improve accuracy, the script first summarizes the raw email content before the main extraction step. A secondary set of rule-based functions refines the LLM's raw output by cleaning job titles and overriding/validating the status based on highly reliable keywords (e.g., checking for "job offered" before accepting a less certain LLM prediction). The final structured data is then compiled into a Pandas DataFrame and saved to Excel.

RAG Agent Logic and Functionality

Knowledge Retrieval

RAG Agent answers user queries efficiently and accurately.

Logic and Functionality

Built a Retrieval-Augmented Generation (RAG) system using LangChain and LangGraph to answer specific questions about job application emails.

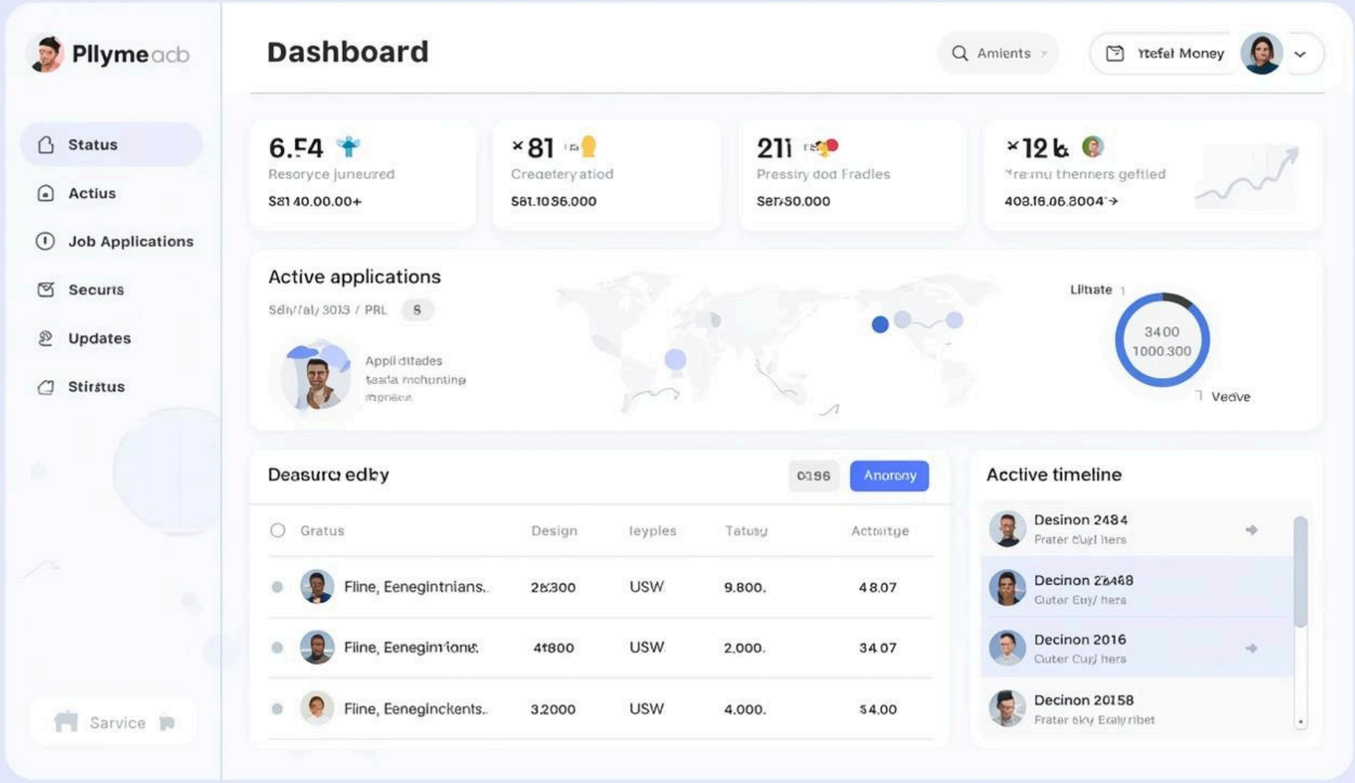
The process first embeds structured job email content (from a pre-processed Excel file) using the powerful BGE-Largemodel and stores it in a Chroma vector database.

The core functionality is orchestrated by a LangGraph workflow with two nodes:

1. Retrieve: Fetches the top 4 most semantically similar emails based on the user's question from the Chroma DB.
2. Generate: Passes the retrieved email context, metadata, and the original question to the local LLM (Ollama Llama 3.1), which generates a concise, factual status update.

This ensures the LLM's answers are grounded only in the user's private email data, preventing fabrication.

Streamlit Interface Overview



User Interface

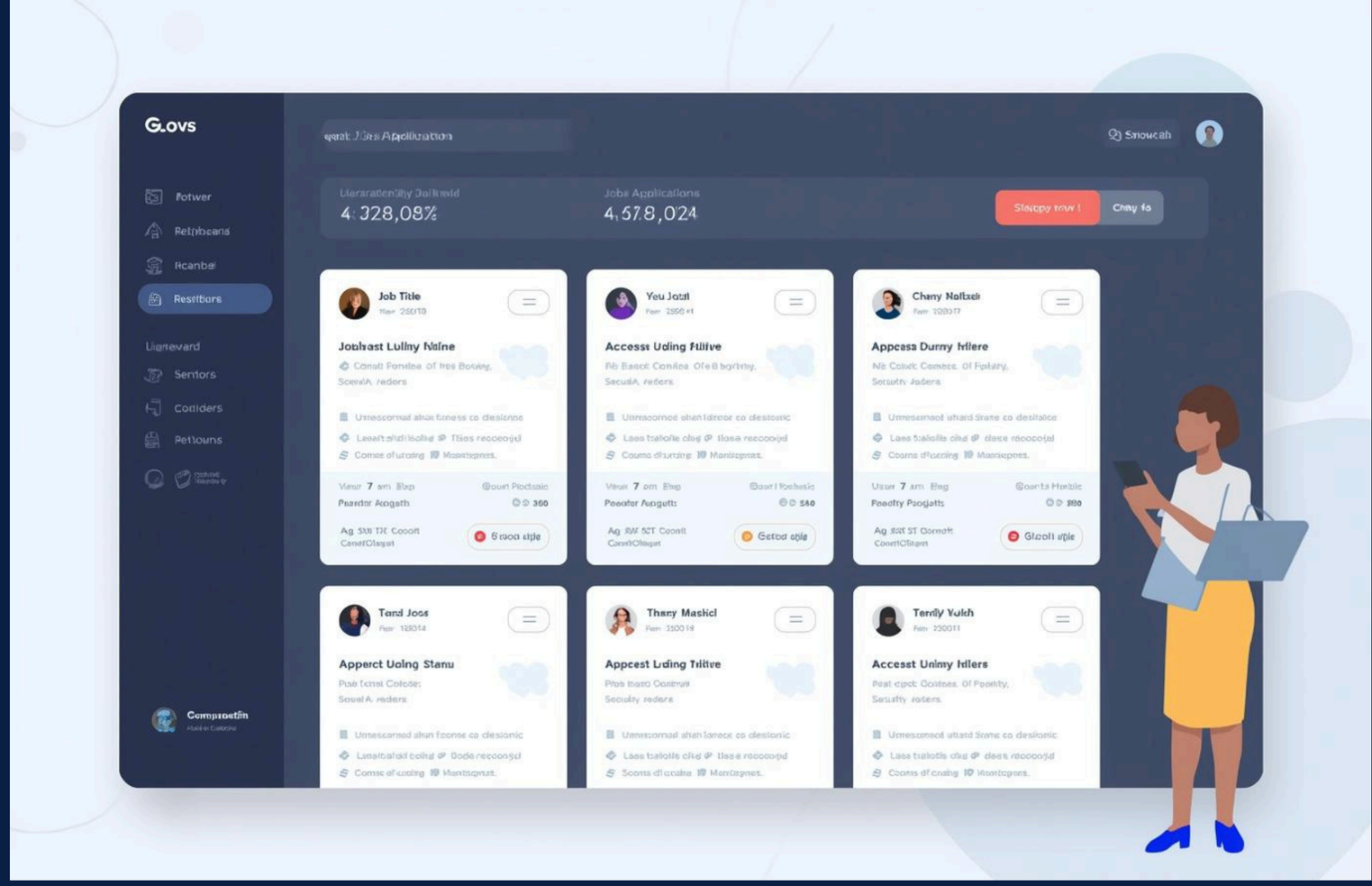
Provides a seamless experience for tracking job applications.

Logic and Functionality

The streamlit application organizes the workflow into three distinct tabs:

1. Fetch + Classify: Triggers the one-time Gmail fetch and runs the job/non-job classifier, displaying raw and classified email datasets and status counts.
2. NER Parsed Jobs: Executes the NER parser to extract structured data (Company, Position, Status) from job emails and displays the resulting clean, parsed records.
3. RAG Assistant: Provides a chat interface that calls the rag.ask(question) function, allowing the user to query their job application data using the local LLM without writing code.

Final Project Outcomes and Benefits



Key Benefits

Users can effortlessly track application status

Thank you for your attention and support!