**Solution Architecture – Resume Bot**

**1. Introduction**

* **Purpose of the App:**  
  An AI-powered bot that automatically reads job descriptions (JDs) and crafts customized resumes and email responses.
* **Target Users:**  
  Job seekers who want to apply faster and smarter.
* **Business Value:**  
  Saves time, improves chances of selection, ensures well-structured responses.

**2. High-Level Architecture**

**Components:**

1. **Frontend (Streamlit)**
   * User interacts with a simple web-based UI.
   * Inputs: Job description, candidate details.
   * Outputs: Suggested email, tailored resume text.
2. **Backend (Python Services)**
   * Logic to parse JD, match with resume, generate response.
   * Calls LLM (e.g., GPT model) for drafting professional replies.
   * Manages session state (chat history).
3. **Desktop Wrapper (Electron)**
   * Wraps the Streamlit app into a desktop application (.exe).
   * Auto-starts backend and displays UI inside a native window.

**3. Development Approach**

* **Frontend** → Built with **Streamlit** for rapid prototyping and interactive UI.
* **Backend** → Python logic + API integration for resume tailoring and email generation.
* **Packaging Options:**
  + **PyInstaller** → Create .exe for standalone use.
  + **Electron** → Wrap into a desktop app (no browser needed).
* **Version Control** → GitHub for code management.
* **Deployment Options** → Can run as:
  + Local desktop app (offline).
  + Web app (if hosted on cloud).

**4. Key Functionalities**

✅ **Job Description Parsing** – Extracts key skills & requirements.  
✅ **Resume Tailoring** – Highlights relevant skills & experiences.  
✅ **Email Drafting** – Generates professional responses dynamically.  
✅ **Conversation History** – Maintains chat-like UI (Streamlit session state).  
✅ **Multi-platform Support** – Works as a desktop app (Electron) or web app (Streamlit).

**5. Solution Flow**

1. User enters **job description**.
2. App processes JD → extracts keywords.
3. Backend tailors **resume content**.
4. Backend generates **customized email reply**.
5. Response shown in **Streamlit UI**.
6. Electron wrapper (if used) → displays it as a desktop app.

**6. Technology Stack**

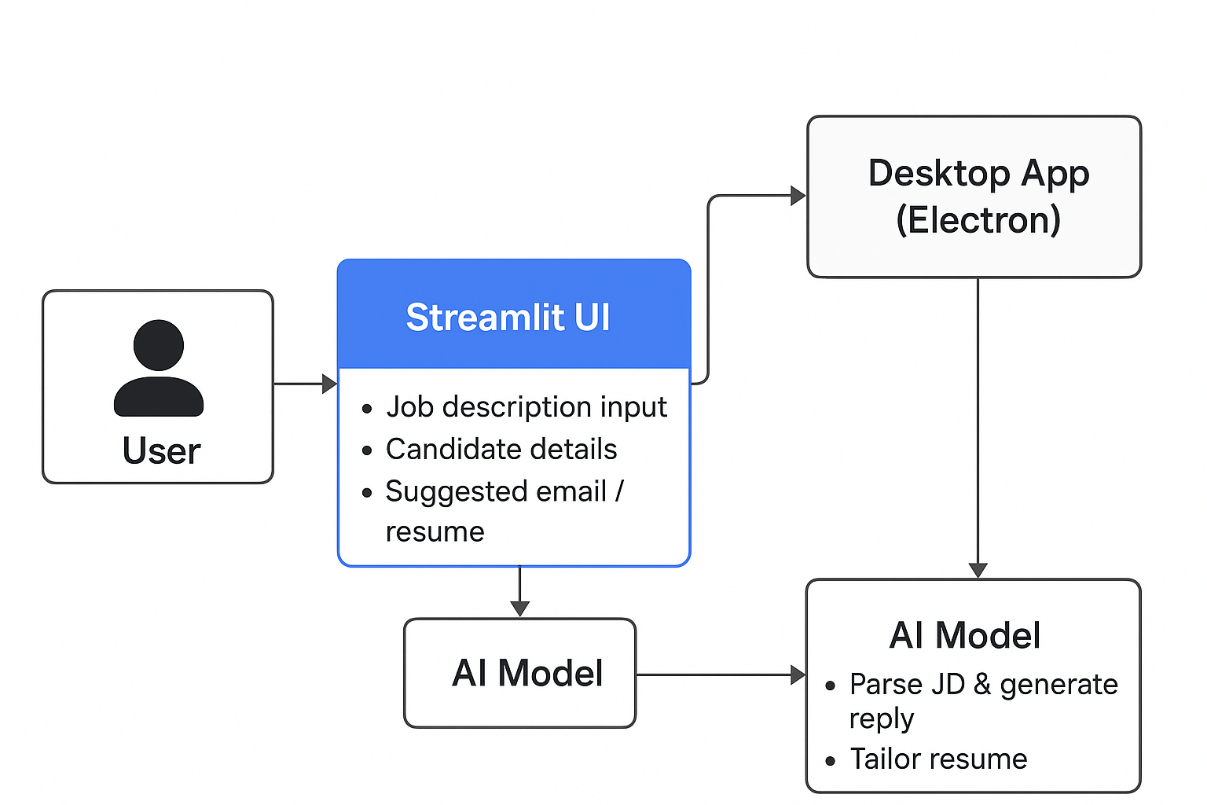
* **Frontend:** Streamlit
* **Backend:** Python (requests, pandas, etc.)
* **LLM / AI API:** OpenAI GPT (or custom model)
* **Packaging:** PyInstaller / Electron
* **Language:** Python 3.10+, JavaScript (for Electron)

**7. Benefits**

* 🚀 Faster job application process
* 🎯 Tailored resumes improve recruiter attention
* 🖥️ Easy-to-use desktop interface (no browser dependency)
* 🔒 Works offline (with PyInstaller + Electron packaging)

You can present this as a **diagram** (boxes + arrows):

* **User → Streamlit UI → Backend (Python) → AI Model → Output → Desktop App (Electron)**



# AI Models Used (Hugging Face)

1. **Text Generation Model**
   * Example: tiiuae/falcon-7b-instruct, mistralai/Mistral-7B-Instruct-v0.1, or google/flan-t5-large.
   * **Role:**
     + Generate professional **email drafts**.
     + Tailor **resume content** based on job description.
2. **Embedding / Sentence Similarity Model**
   * Example: sentence-transformers/all-MiniLM-L6-v2 or bert-base-nli-mean-tokens.
   * **Role:**
     + Extract and represent **skills & keywords** from job description.
     + Match candidate experience → job requirements (semantic similarity).
3. **Summarization / Keyword Extraction Model (optional)**
   * Example: facebook/bart-large-cnn or t5-base.
   * **Role:**
     + Summarize long job descriptions.
     + Extract important phrases & responsibilities.

# 📌 Summary in Your Context

* **Frontend (Streamlit)** → takes JD + candidate details.
* **Backend (Python)** → sends text to Hugging Face models.
* **Models** →
  + **Embedding model** finds relevant skills.
  + **Generation model** drafts email + resume bullet points.
  + **Summarization model** (if used) shortens JD before processing.
* **Output** → professional email + tailored resume shown in UI.

Bottom of Form