# Social Support AI with LangGraph Agents

**Solution Summary:** This project for which code is made available at [Social-Support-AI-Application](https://github.com/Creattorr/Social-Support-AI-Application.git%20) on GitHub automates eligibility assessments for government social support applications using Machine Learning and agent-based orchestration with LangGraph.

1. Problem Overview  
The manual processing of social support applications causes delays, inconsistencies, and human biases. This solution streamlines the end-to-end workflow through AI-driven automation, including document ingestion, validation, eligibility classification, and recommendations via a locally hosted multimodal system.

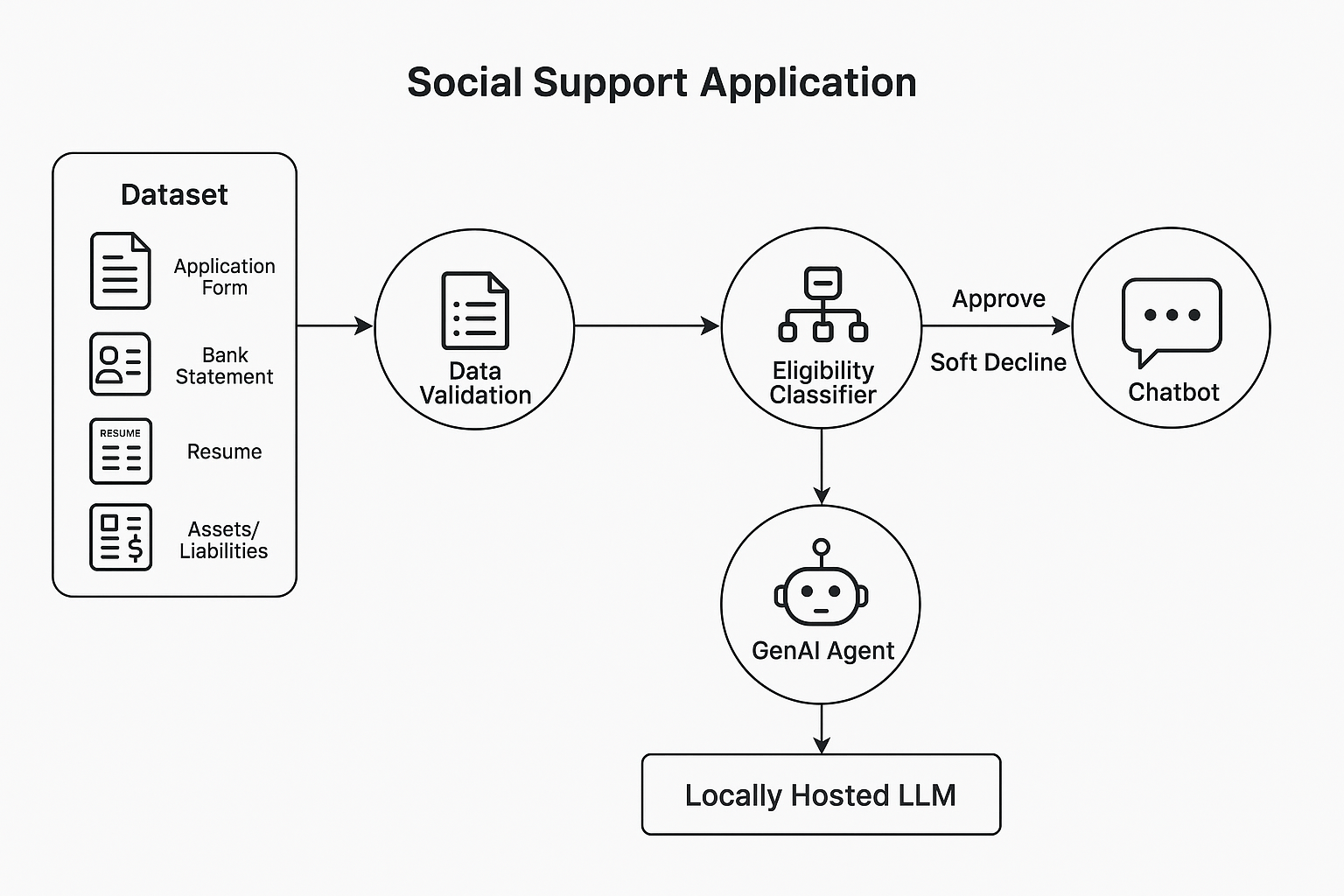
2. High-Level Architecture  
The architecture integrates:  
- Multimodal Inputs: PDFs, images, Excel, scanned forms  
- OCR + Preprocessing: Tesseract, pdf2image  
- Agentic Pipeline: Data Extraction → Validation → Eligibility → Recommendation  
- ML Engine: Scikit-learn Random Forest classifier  
- LLM Agent: LangGraph agent for reasoning with user context  
- API Backend: FastAPI  
- Frontend: Streamlit  
- Storage: Local file system, in-memory data processing

## 3. Tools Justification

|  |  |  |
| --- | --- | --- |
| **Component** | **Tool** | **Rationale** |
| OCR | Tesseract | Lightweight, effective open-source OCR engine |
| ML | Scikit-learn | Robust classical models, easy deployment |
| LLM Hosting | LangGraph + local model (Ollama) | Privacy-preserving, customizable reasoning framework |
| Backend | FastAPI | Fast, async support, modern Python standards |
| Frontend | Streamlit | Rapid development and interactive UI |
| Version Control | GitHub | Collaboration, CI/CD readiness |

4. Modular AI Workflow  
- **Ingestion**: Accepts documents via Streamlit UI or API.  
- **Extraction Agent**: Uses regex + OCR to convert inputs to structured text.  
- **Validation Agent**: Checks for field consistency and completeness.  
- **Eligibility Agent**: Extracts features and uses ML model for decisioning.  
- **Recommendation Agent**: Provides final suggestion (approve/decline/upskill).  
- **LLM Agent**: LangGraph-powered for natural language clarification and dynamic reasoning.

The architecture of workflow is as below:



Folder structure of code is as below starting with root directory:

**Social-Support-AI-Application** /

│

├── data/ # Synthetic + Uploaded data files

│ ├── application\_form.csv

│ ├── credit\_report.csv

│ └── ...

│

├── ingestion/ # File parsers and extractors

│ ├── parse\_pdf.py

│ ├── parse\_excel.py

│ └── ocr\_emirates\_id.py

│

├── validation/ # Data validation logic

│ └── validate\_data.py

│

├── eligibility\_model/ # ML model code

│ ├── train\_model.py

│ ├── model.pkl

│ └── features.py

│

├── agents/ # GenAI agents and orchestrator

│ ├── master\_orchestrator.py

│ ├── data\_agent.py

│ ├── validation\_agent.py

│ ├── eligibility\_agent.py

│ └── recommendation\_agent.py

├── api/ # FastAPI endpoints for serving

│ ├── main.py

│ └── routes.py

│

├── ui/ # Streamlit front end

│ └── app.py

│

├── utils/ # Shared utilities

│ ├── logger.py

│ └── config.py

│

├── requirements.txt

├── architecture.png # Final architecture diagram

└── README.md # Run instructions & setup

5. Future Improvements  
- Integration with government APIs for real-time document verification.  
- Add biometric and image recognition for ID documents.  
- Expand ML features (family size, expenses, asset data).  
- Integrate LangSmith or Langfuse for observability.  
- Add user authentication and data encryption.

6. API & Data Pipeline Considerations  
- Modular, stateless FastAPI endpoints  
- Async processing for file uploads  
- Use of feature engineering for consistent ML input  
- Optional vector storage (ChromaDB) for semantic document lookup

7. Scalability & Security  
- Stateless backend allows horizontal scaling  
- Local LLM avoids external data sharing  
- Codebase structured for CI/CD pipeline readiness

8. Deployment Instructions  
- Clone GitHub repository  
- Install dependencies: `pip install -r requirements.txt`  
- Run backend: `uvicorn app.api.main:app --reload`  
- Launch UI: `streamlit run frontend/app.py`  
- Train model: `python scripts/train\_model.py`