FinSaathi: AI/ML Implementation & Hackathon Strategy Guide

Project: AI-Powered Personal Finance Assistant for Underserved Communities **Theme:** Ignite the Future (Hack-A-League 4.0)

1. The AI/ML Arsenal (What We Are Building)

Component	Selected Model	Purpose in FinSaathi	Implementation Method
Main Brain / RAG	**Llama-3.1-8B-Instruct**	Reasoning, budgeting advice, and answering financial queries based on RBI/SEBI docs.	Hosted locally via **Ollama**. Uses LangChain to pull context from ChromaDB.
Multilingual Layer	**Airavata (AI4Bharat)**	Providing native, culturally accurate responses in 12+ Indian languages (Hindi, Tamil, etc.).	**QLoRA Fine-Tuning** via `SFTTrainer` on the **Bactrian-X** dataset subset.
Agentic Action Engine	**Mistral-7B-v0.3**	Triggering background tasks (e.g., updating a budget, fetching live gold prices).	**Native Function Calling** / MCP (Model Context Protocol).
Advisor AI Clone	**Llama-3.1 + ChromaDB**	Mimicking a human financial advisor's specific advice style when they are offline.	**Few-shot Prompting + RAG** using historical logs of the advisor's past answers.
Content Generator	**Gemma-2-9B**	Creating the bite-sized daily financial literacy nudges.	Zero-shot prompting (Off-the-shelf, no training required).
Predictive Analytics	**Prophet (Meta)**	Forecasting month-end spending based on the first week's transaction patterns.	Python time-series script triggered via Fastify/Redis.
Fraud/Anomaly Detection	**Isolation Forest**	Flagging unusual spending behavior (e.g., 3x normal food budget) as a soft fraud check.	`scikit-learn` unsupervised model trained on user transaction history.
SMS/Transaction NLP	**DistilBERT**	Extracting Merchant, Amount, and Category from bank SMS or manual text entry.	**Supervised Fine-Tuning (SFT)** on `BANKING77` and synthetic SMS logs.

2. Backend Infrastructure & Orchestration

To support a multi-agent, heavily ML-driven platform without crashing during a demo, your backend must be stateless and fast.

- **Primary Framework: Fastify (Node.js/TypeScript)**. Extremely high throughput, ideal for streaming LLM tokens back to the client.
- **Database (Relational): PostgreSQL (via Prisma)**. Stores User Profiles, Transaction Histories, Savings Goals, and RBAC (Casbin) policies.
- **Database (Vector): ChromaDB**. Stores the embeddings of financial literacy PDFs and the "Golden Logs" of the human advisors for the AI Clone.
- **AI Serving: Ollama**. Runs your Llama 3.1 and Mistral models locally, exposing a clean REST API for Fastify to consume.
- **Asynchronous Queue: Redis + BullMQ**. Handles background ML tasks like running the Prophet forecasting script or the Isolation Forest anomaly checks without blocking the main chat UI.

3. Advanced Implementation: The Advisor Clone

Instead of trying to train a new model for every single advisor (which takes too much time and compute), we will use a **RAG-based Persona Injection** technique.

The Workflow:

1. **Log Collection:** Every time a human advisor answers a query, Fastify saves `{advisor_id, user_query, advisor_response}` to PostgreSQL and embeds it into ChromaDB.
2. **Offline Trigger:** When a user asks a question and their assigned advisor is marked as `Offline`, the request is routed to the "Clone Engine".
3. **Context Retrieval:** ChromaDB fetches the top 5 most relevant past answers given by that specific `advisor_id`.
4. **Prompt Synthesis:** Fastify sends a prompt to Llama-3.1 via Ollama:

> *"You are the AI Clone of Advisor [Name]. Your advice style must match these past examples: [Insert ChromaDB Context]. Answer the following user query in their preferred language: [User Query]."*

4. The 24-Hour Hackathon Strategy (Best Advice)

Attempting to fine-tune and integrate 8 different machine learning models in 24 hours is a recipe for disaster. To win, you need the **illusion of scale** backed by **one deeply technical centerpiece**. Here is how to execute this strategically:

Rule 1: Only Fine-Tune ONE Model (The Centerpiece)

- **What to do:** Focus all your training time on fine-tuning the **Multilingual Layer** (e.g., fine-tuning Llama or DistilBERT on a small subset of Bactrian-X/Banking77).
- **Why:** Judges look for "Technical Excellence". Showing a Jupyter notebook where you successfully executed QLoRA on a consumer GPU is impressive.
- **The Hack:** For everything else (Gemma, Mistral, Llama 3 for RAG), **use the base models directly via Ollama** without fine-tuning them. Prompt engineering is your friend here.

Rule 2: Hardcode the Classical ML (If you run out of time)

- **What to do:** Prophet and Isolation Forest are great, but dealing with Pandas dataframes and Python microservices in Hour 20 of a hackathon causes bugs.
- **The Hack:** Write the Python scripts, but for the live demo, have a **synthetic dataset** pre-loaded in Postgres that perfectly triggers the "Anomaly Detection" nudge on cue. Do not rely on live data entry to trigger your edge cases during a 3-minute pitch.

Rule 3: Stream Everything

- **What to do:** Ensure your chat interface streams tokens one by one (Server-Sent Events) rather than waiting for the whole response.
- **Why:** Even if Ollama is running slightly slow on your local machine, streaming makes the app *feel* incredibly fast and premium, which aligns perfectly with your "Liquid Glass" UI philosophy.

Rule 4: Master the MCP (Model Context Protocol)

- **What to do:** Your biggest "wow" factor isn't just chatting; it's the AI *doing* things.
- **The Hack:** Spend Hour 9-13 strictly wiring up **Mistral-7B** to trigger your Fastify API endpoints. When the user types "Log 500 rupees for chai", the LLM should output a JSON payload that updates the database and instantly refreshes the React dashboard.

Rule 5: UI is King in the Pitch

- **What to do:** Your "Liquid Glass" design system is your secret weapon.
- **The Hack:** An AI that gives decent advice in a breathtaking, smooth, glassmorphic UI will score higher than a perfectly fine-tuned AI in a broken, unstyled HTML page. If you fall behind schedule, **protect the UI team's time** at all costs.